UNIVERSITÄT ZU LÜBECK
INSTITUTE OF COMPUTER ENGINEERING

Platooning of Autonomous Cars as a Multi-Armed Bandit Problem

Platooning im autonomen Fahren als Multi-Armed Bandit Problem

Master's Thesis
in Computer Science at University of Lübeck

written by
Julian Schwarzat

Supervised by
Prof. Dr.-Ing. Heiko Hamann

Lübeck, May 19, 2020

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne die Benutzung anderer als der angegebenen Hilfsmittel selbstständig verfasst habe; die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe des Literaturzitats gekennzeichnet.

Lübeck, 19. Mai 2020

Danksagung

Hiermit möchte ich mich bei allen bedanken, dich mich während der Anfertigung meiner Masterarbeit unterstützt haben.

Speziell gilt mein Dank

- Prof. Dr.-Ing. Heiko Hamann für die Bereitstellung dieses interessanten Themas, die ständige Hilfsbereitschaft und die angenehme Betreuung.
- allen Mitarbeiten und Studenten des Instituts für Technische Informatik, in dem ein sehr angenehmes Arbeitsklima herrscht.
- meinen Korrekturleserinnen und -lesern Marius Krusen, Christin Reher, Luca Wilke und Malte Eilhardt.
- Michael Werner, der mir bei Fragen rund um LaTeX und TikZ zur Seite stand, bei denen ich sonst verzweifelt wäre.

Abstract

Autonomous vehicles open up new opportunities. One of them is the idea of platooning, where multiple autonomous cars drive in close proximity in order to form a road-train. Platooning promises several benefits. There is a significant increase in safety. Slipstream effects can save up to 20 % of fuel consumption [9]. Our main objective is to increase the total efficiency across all cars by allowing a dynamic reconfiguration of platoons at runtime. We use methods of machine learning that were developed to solve the problem of multi-armed bandits. It is the problem of choosing from uncertain options while maximizing expected gain. We implement a simulation based on the open source software PLEXE / SUMO [27]. We implement five algorithms, ε-greedy, UCB 1, Bayes UCB, Thompson-Sampling and the approach of Heinovski and Dressler [14], and compare them in different scenarios. We focus on two highway scenarios, the decision time of each algorithm, an estimated minimal profit for platoon switches and a initially speed up for faster platooning at start. In dynamic scenarios, the algorithms do not significantly differ. Choosing a conservative scenario leads to fewer changes and significantly increases the results of ε-greedy, UCB 1, Thompson-Sampling algorithms. The success of platoon formation is highly dependent on the chosen parameters mentioned above. A variety of parameter researches remain open. The parameter *decision threshold* has a great influence how fast vehicles form platoons. Further work may focus on a dynamic threshold in dependence of the current vehicle density or on acknowledgment-based inter-vehicle communication.

Kurzzusammenfassung

Autonomes Fahren bietet eine Vielzahl neuer Möglichkeiten. Eine davon ist das sogenannte Platooning, bei dem mehrere autonome Fahrzeuge mit minimalem Abstand zueinander fahren, um aus Verkehrssicht eine Einheit zu bilden. Platooning bietet mehrere Vorteile. Neben einer deutlichen Erhöhung der Verkehrssicherheit kann durch Fahren im Windschatten bis zu 20 % vom Kraftstoffverbrauch eingespart werden [9]. Unser Ziel ist es die Gesamteffizienz von Fahrzeugen zu steigern, indem wir zur Laufzeit dynamische Neukonfigurationen der Platoons zulassen. Wir verwenden Methoden aus dem maschinellen Lernen, die auf das Lösen des sogenannten Multi-Armed Bandit Problems ausgelegt sind. Dieses Problem handelt von dem auswählen aus einer Vielzahl an unsicheren Möglichkeiten, während wir versuchen, den Gewinn zu maximieren. Wir implementieren eine Simulation auf Basis der Open Source Software PLEXE / SUMO [27]. Wir implementieren fünf Algorithmen, ε-greedy, UCB 1, Bayes UCB, Thompson-Sampling und den Ansatz von Heinovski and Dressler [14] und vergleichen diese in verschiedenen Szenarien. Wir konzentrieren uns auf zwei Autobahnszenarien, die Zeit, die für die Entscheidungen eines Algorithmus zur Verfügung steht, einen variablen Schwellwert für eine minimale Verbesserung, den ein Platoonwechsel erbringen muss und auf eine anfängliche Beschleunigung für ein beschleunigtes Platooning zu Beginn. In dynamischen Szenarien unterscheiden sich die Algorithmen nicht wesentlich. Die Wahl eines konservativen Szenarios führt zu weniger Änderungen und erhöht die Ergebnisse der Algorithmen ε-greedy, UCB 1 und Thompson-Sampling erheblich. Der Erfolg der Bildung von Platoons hängt stark von den oben genannten gewählten Parametern ab. Eine Vielzahl von Parameteruntersuchungen bleibt offen. Die Zeit, die einem Fahrzeug zwecks Entscheidung zur Verfügung steht, hat einen großen Einfluss auf die Geschwindigkeit der Platoonbildung. Weitere Arbeiten können sich auf eine dynamische Schwelle dieser Zeit in Abhängigkeit von der aktuellen Fahrzeugdichte oder auf eine ackknowledgment-basierte Kommunikation zwischen Fahrzeugen konzentrieren.

Contents

1 Introduction

In recent years, road traffic management has become increasingly important. Infrastructure that was not designed for the mass of vehicles. We have to deal with serious consequences [5]. Increased traffic also leads to an increased risk of congestion. This is accompanied by a rise in pollution as well as stress-related problems of continuously affected humans. In Germany, railways have repeatedly been considered as a high-priority alternative in recent years. However, road traffic has clearly been given preference.

With recent advances in technology, the idea of autonomous driving is becoming more and more realistic. The foundation has been laid less than 30 years prior [28]. Currently autonomous systems are being tested in various areas and such systems will become an integral part of everyday life in the near future. The vision of future road traffic consisting mostly of autonomously driven vehicles opens up many new opportunities. One of them is the idea of platooning, where multiple autonomous cars drive in close proximity in order to form a road-train [1].

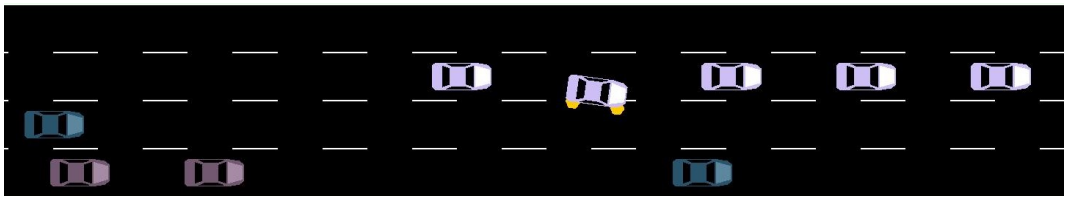

Figure 1.1: Example of platoon formation on a highway. Gray vehicles are single vehicles. Same colored cars are driving in the same platoon. Dynamic reconfiguration of platoons leads to vehicle switches. The fourth white vehicle has detected a better fitting platoon and has started a platoon switching process.

Platooning promises several benefits. There is a significant increase in safety. From the year 2000 until 2019, the annual mortality rate in road traffic in Germany has more than halved from about 7.500 to 3.500 killed people [30]. This trend goes hand in hand with driver-assistance systems becoming more popular in vehicles. Nowadays, drivers' inattention leads to 18 % of total fatalities in traffic [25]. In a platooning road-train, drivers are relieved by handing over control to the leading car, improving their comfort. Depending on the scenario, the first car is either controlled by a trained driver or drives completely autonomously [1]. A typical scenario for a trained driver is a truck-road train. A company is delivering freight. The first truck is controlled by a driver. The other trucks are following this driver autonomously. By not automating the control of the leading car, this technology is much easier to realize. Another main improvement of platooning is the increasing traffic flow. Autonomous vehicles, which are organized in platoons, can maintain a desired speed and thus improve traffic flow rate without much deceleration or acceleration. The danger of traffic jams, which are nowadays often caused by unpredictable driving behavior of humans, is significantly reduced. This is accompanied by increased productivity and economic growth. Employees could reach their workplaces faster, the duration of business trips as well as the transport time of goods is shortened and due to driving multiple trucks with one or even no driver, forwarding companies may save staff costs [15]. Finally, there are also environmental benefits. A consistent driving style reduces CO_2 emissions. Due to increased safety, the actual safety margin can be reduced. Therefore slipstream effects are gaining in importance, which can save up to 20 % of fuel consumption [9].

1.1 Related Work

First platooning experiments were carried out as early as 1997 as part of the PATH project in the USA. The goal was to produce a significant increase in the capacity of a highway lane with a minimum of new infrastructure construction [28]. Although technically not possible at the time of research, the researchers assumed that all vehicles were completely autonomous. Around 2008, more projects followed in different regions of the world, such as Energy ITS (2008, Japan) [31], GCDC (2011, Netherlands) [1], and SATRE (2012, Europe) [25]. These approaches can be roughly subdivided into two groups. On the one hand, projects (Energy ITS, PATH) focu-

sed on Truck Platooning, mainly due to the study of energy-saving effects caused by slipstreams. On the other hand, there were projects (SATRE, GCDC) dealing with the research of mixed vehicle platoons. For example, the objectives of SARTRE were to develop strategies and technologies that allow the operation of vehicle platoons in normal highway operation [1].

The cars' sensors, inter-vehicular communication, and Cooperative Adaptive Cruise Control (CACC) are the tools to control a platoon. An open challenge for platooning is how to form platoons initially starting from unorganized traffic. Such a multiagent system can be controlled in several ways. These are subdivided into centralized and distributed approaches [14]. For an individual car this includes different aspects of a decision-making mechanism. There is already research about these mechanisms. L.H.X. Hobert has collected various aspects in his master thesis in chapter 3.1 [15]. Depending on the prioritization, there are various platooning methods. This includes, platoons that are assembled according to their destination. Agents in this case join the platoon only if their target is closer or identical to the destination of the platoon. Heinovski and Dressler present a distributed approach to the initial creation of platoons and compare their performance with central approaches [14]. Therefore, assumptions are made that one platoon must not join another and that a car cannot change the platoon. In order to allow the agents to switch between platoons, Heinovski presented a method with so-called feasible and unfeasible platoons in his master's thesis [13]. If the characteristics of a platoon match the requirements of an agent, he speaks of a feasible platoon. If an agent cannot detect a feasible platoon, he joins an unfeasible platoon for the time being and then switches to another feasible platoon as soon as detected. However, feasible platoons are also different and from a single car's point of view they can be more or less beneficial. Due to different speeds, the possibilities of joining a feasible platoon are varying over time. This can lead to situations, where an agent is no longer in a position to join a possibly better matching but initially undetected platoon, because it has already joined a feasbile platoon.

All approaches mentioned so far are regarded as static. This means, that agents cannot leave a platoon in general, after they have joined one. This restriction limits the complexity of the problem, but at the same time leads to undesired effects.

For example, in their decentralized approach, Heinovski and Dressler [14] have an average platoon size of 2.7 members. The optimal case includes larger platoons and is not attainable with a static approach. A dynamic reallocation of platoons during runtime is requested. Therefore, various cases must be considered. Thus, a scenario is conceivable in which the effort of frequently changing agents neutralizes the benefits of platooning. The behavior of a dynamic system for forming platoons remains unclear in literature.

1.2 Approach

Our main objective is to increase the total efficiency across all cars by allowing a dynamic reconfiguration of platoons at runtime. Static platoon formation (i.e. once at the beginning of an experiment) will serve as the base line for comparison. We use methods of machine learning that were developed to solve the problem of multi-armed bandits. The problem is how to choose from several options in order to maximize the expected gain. We do only partially know the profit of each option. Key is to balance the exploration-exploitation trade-off [12]. We take a closer look at the following scenarios.

- **Start to End Scenario:** This is a highway scenario, where all vehicles start at the beginning and continuously drive until the end of the highway. This is expected to be the more conservative scenario because we do not have vehicles entering or leaving the highway in between start and end.
- **Dynamic Scenario:** In this scenario the highway is separated by highway entries and exits. These exits occur in constant intervals. At each exit vehicles may leave or enter the highway. Already formed platoons are disrupted and have to reorganize. This is expected to be a highly dynamic scenario with a lot of platoon switches.

Among these scenarios, we test several parameter setups. We analyze related work and extract important parameters for our work in order to create these setups. The objective is about how specific parameters influence the overall system, thus influencing the platoon formation process. Finally, we compare our results with existing approaches. Here, we use the approach of Heinovski and Dressler [14] as a baseline.

1.3 About this work

This work is separated into the following parts. In chapter 2 we introduce work-related scientific knowledge. This includes a general understanding about autonomous driving, vehicle-to-X communication and the concepts of platooning as well as basic knowledge about the multi-armed bandit problem and machine learning algorithms we want to adopt to our system. Finally, we introduce our choice of simulation software PLEXE. In chapter 3 we start with an analysis of existing distributed platoon forming approaches. This delivers information we use to setup our own model. The model is implemented in PLEXE. Afterwards we adopt the multi-armed bandit approach as well as machine learning algorithms to our system. The results in chapter 4 are separated into the categories quality of implementation, analyses of the model setup with different parameters, system stability and the possibility of platoon disruption. This concludes in a comparison of multiple different scenarios. Our algorithms are tested in both scenarios. These results are interpreted in chapter 5. We discuss the influence of our chosen parameters, the model setup in regard to implementation and stability and the comparison of our algorithms in multiple scenarios. Finally, chapter 6 summarizes the quintessence of our work and gives a prospect about further topics to work on.

2 Fundamentals

First of all, we start with an overview about the state of the art in autonomous driving and its implications to daily life in section 2.1. To establish platooning we require a vehicle to vehicle communication system. Section 2.2 deals with this so called V2X communication (vehicle to X). We go on with the concept of platooning in section 2.3. Our approach is based on a distributed scenario. Besides that, of course there are centralized approaches in literature. Section 2.4 gives an overview about the ideas of these fundamentally diverse approaches. The objective of this thesis is to adapt the multi-armed-bandit problem to platooning. This opens up a lot of algorithms from the machine learning sphere. Section 2.5 gives basic knowledge about the multi-armed-bandit problem. Subsequent section 2.6 deals with machine learning methods, working great with the multi-armed-bandit problem. Finally, we need a testbed for our work. We need a detailed, well-working simulation software. Our choice is the package PLEXE from Segata et al. [27]. The last section 2.7 introduces this software.

2.1 Autonomous Driving

Autonomous driving is currently a popular and ubiquitous topic. Almost everyone is informed by the media about this topic on a basic level. The term autonomous driving is a collective term for the movement of vehicles that behave largely autonomously [20]. In the public discussion, this primarily includes road and rail traffic scenarios. A milestone in this development is the autonomous driving style of two subway lines in Nuremberg since 2008 [21]. Another focus of research is the application in logistics. A well-known example is the container terminal Altenwerder in the port of Hamburg, which is regarded as state of the art worldwide due to its high degree of automation [11]. The Federal Highway Research Institute of Germany divides developments in autonomous driving into different degrees of automation [4].

- **Level one: Driver only.** The driver is responsible for all matters, especially for the longitudinal and transverse guidance of the vehicle.
- **Level two: Driver's assistance.** The driver performs either the longitudinal or transverse guidance of the vehicle permanently. The other one is performed by the system. Main conditions are that the driver monitors the system permanently and can take complete control of the vehicle at any time.
- **Level three: Partially automated.** The system takes over the lateral and / or longitudinal guidance over a longer period of time or in certain situations. Again, the driver must supervise at all times and be ready to take over the control of the vehicle.
- **Level four: Highly automated.** The biggest difference to partially automated systems is the required behavior of the driver. While the system performs the same tasks as before, the driver no longer has to monitor it. The system can react adequately to any situation with the objective of minimizing the risk. If necessary, the driver is asked to take over control with sufficient preparation time.
- **Level five: Fully automated.** In this scenario, a completely autonomous driving style of the vehicle is assumed. However, in certain situations it is required for the driver to take over control. If this does not happen, the system can react and leads the car to a state of minimal risk.

As it is popularly understood, the line between manual and autonomous driving is the partially automated system. In certain situations, this is already approved and increasingly common, such as automated parking. In today's state of the art, vehicles in the highly automated category are feasible. Various automotive companies test their systems under real world conditions [20]. The safety of vehicle occupants already reaches a higher level than ever achieved by humans. Still problematic are the ethical, legal and social implications (ELSI) of autonomous driving. With the handing over of responsibility to the car, questions arise that also become urgent in many other areas of human-machine interaction as technology advances. Cars must make ethical decisions that have not yet been sufficiently discussed. There is also the question of reliability and data security. Even if the probability of failure

is significantly less than human error, the question of legal responsibility must be clarified in the case of technical failure. Malicious external systems must not have the opportunity to take control. Data, such as location, speed, starting location and destination, are recorded by the system but must remain private. Furthermore, law needs to be adapted. In Germany, the Road Traffic Act 2017 was amended, so that the driver no longer has to devote attention to traffic as long as the car is in autonomous driving mode. However, the driver must always be ready to intervene, should the situation require that. In pilot projects, fully autonomous cars must always be manned by a driver who can intervene in an emergency situation [16].

A study commissioned by the General German Automobile Association estimates the market-ready introduction of fully autonomous driving for 2030. This means that cars can be completely autonomous both on motorways and in urban traffic and require no driver intervention. Larger numbers of autonomous vehicles will not be seen on the streets until 2040. This is mainly due to the long service life cycle of a car averaging 15 to 20 years. As a result, new technology typically has a specific time delay [23].

2.2 V2X Communication

Vehicle-to-X communication is the collective term for a lot of different communication systems in transportation systems. This includes vehicles, like cars, airplanes and trains, as well as communication hubs next to the road. The overall objective is to make autonomous driving possible by sharing information and allow motion control. Information sharing as vehicle-to-hub communication is available these days, mainly used for navigation systems, detecting traffic jams and proposing alternative routes. The safety aspect considers warning messages for example about road conditions or breaking processes of leading cars. Besides assisting the driver, information sharing in combination with motion control is the basis of autonomous driving. Getting information from other cars offers services like collision avoidance, runway incursion avoidance and adaptive cruise control. Even more it makes formation planning available, such as platooning.

V2X communication systems must be reliable, fast and scalable. A high reliabi-

lity is crucial for using sensors. Errors may lead to several crashes, injuries and dead people. Also in case of sensor errors, there are mechanisms to detect them, while the system stays reliable. This requires redundancy. Since we have to handle a real-time system, the latency must be small enough to guarantee receiving messages and consequently actions. Last, since typical scenarios for autonomous driving are traffic on highways, the vehicle density is potentially high. A communication system must effort reliability under high workload. The resulting behavior must be deterministic, whether a small or high amount of vehicles are in the system [33].

2.3 Platooning

A platoon is a system, in which several vehicles drive close to each other without violating traffic security [25]. Therefore, the system needs a vehicle control system on the basis of a V2X communication system. This system is currently being researched and developed. Some testings of such a system have been performed in several countries [1]. Intended advantages are lesser fuel consumption on highways due to slipstream and the concentration relief of drivers. In case of a platoon controlled by the lead vehicle, following cars might drive without a driver, thus saving forwarding companies a lot of money. In order to realize such a system, a control system is required. In the following, we introduce the concepts of Cruise Control (CC), *Adaptive Cruise Control* (ACC) and *Cooperative Adaptive Cruise Control* (CACC).

CC is common in a lot of brand-new cars and it was already patented in 1990 by Kakinami et al. [17]. The main achievement is automatic speed handling by the system. The driver selects a desired driving speed, while the system controls the actual speed by accelerating or braking the vehicle. With ACC [24], the driver now choses a headway time to vehicles in front of the car. Next to the desired speed the control system also keeps a constant gap based on the headway time. The system constantly adapts the speed of the leading car. If the distance falls below the safety gap, the car automatically decelerates. Both control systems only allow longitudinal speed guidance. The lateral speed (for example for overtaking maneuvers) can not be controlled by these systems. While ACC allows a single car to follow up a car in front, this concept does not work with platooning. [13].

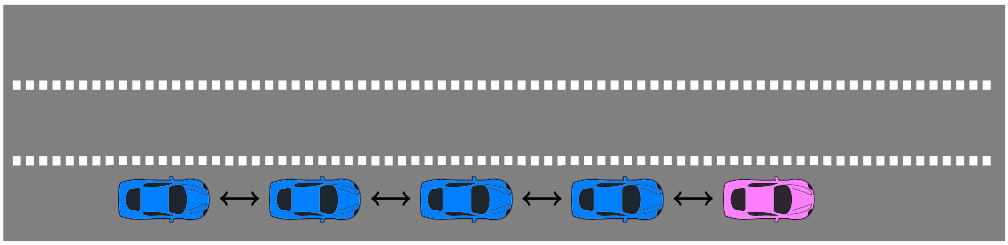

Figure 2.1: An example of a driving platoon. The pink car is the leader. The blue cars are following their leader. The arrows symbolize communication between those cars.

Figure 2.1 shows a driving platoon. The control is organized as CACC, an extension based on ACC. It deals with the problems occurring from ACC driving in a vehicle chain. Now it is possible to perform platooning maneuvers [32]. The red leading car is setting the driving speed and decides whether to start platooning maneuvers. It is the only car controlled by ACC. All platoon members except the leader are using a CACC algorithm. The vehicles inside a platoon communicate with each other in order to exchange information. Each vehicle knows about its leader and it's front car. A general aim of all CACC algorithms is to regulate the speed in relation to the leader's speed and to keep the inter-vehicle safety gap to the predecessor on the same level. The size of the safety gap is determined by the headway time of the platoon leader.

Platoon maneuvers are generally lane changing maneuvers. Figure 2.2 shows a generic overtaking process. The platoon leader detects a single driving vehicle with a lower speed than itself. The leader is the head of the platoon, thus it is able to initiate an overtaking maneuver. Controlled by ACC, the leader closes up to the vehicle in front. The leader communicates with its followers and ask them, if they are able to follow a lane change. Each member can detect other vehicles around them with its own sensors. A leader may not know about other vehicles blocking the lane change for one of its members, but can communicate with them. If the platoon is not able to change lane, the leader adapts the predecessors speed and drive behind the car until there is a chance to overtake. If all members give their okay, the leader starts the maneuver [32]. The members follow the lane change. When all

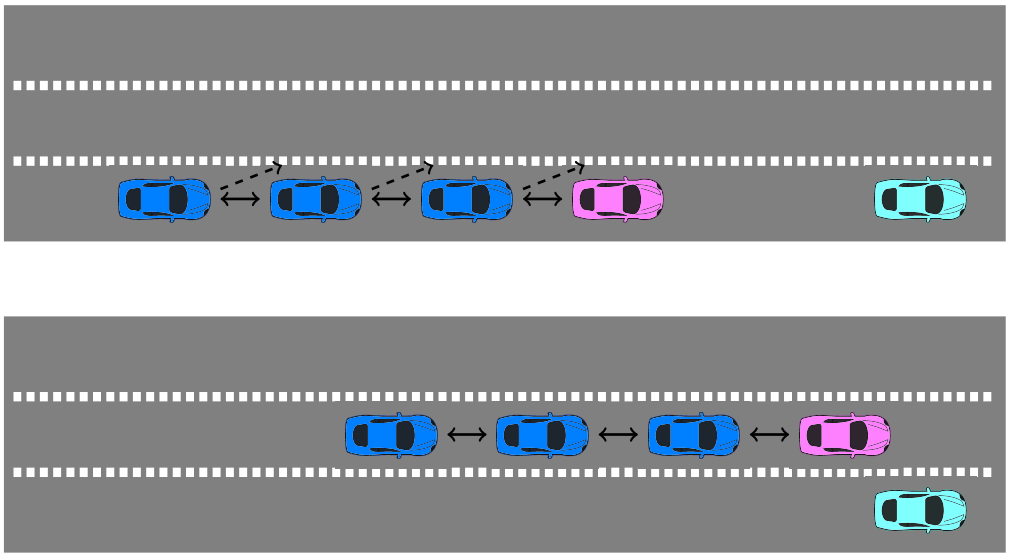

Figure 2.2: Example of an overtaking maneuver as a platoon. The pink car is the leader. The blue cars are following their leader. The cyan car is driving on its own. The arrows symbolize communication between those cars. The dotted arrows symbolize the intention of doing an action.

members have successful completed the lane change, the platoon accelerates to its original speed again. When the last member of the platoon signalizes that it, too, has overtaken the single driving vehicle, the leader initializes another lane changing maneuver. This maneuver is similar.

While platoon-internal maneuvers are possible to handle at a certain degree of complexity, dynamic forming of platoons increases this complexity dramatically. Hobert [15] shows a summary of published platoon forming approaches, but remarks at the same time that there is only little scientific work available on this topic. Based on his summary, the approaches are separated into classes. All classes have in common that they focus on static platoon forming. Also the most flexible approach does not have a member changing the platoon during the drive. The state of the art is far from thinking about joining and leaving processes. Bergenhem et al. [1] give an overview about several platoon field studies. The researchers focus in general in either V2X or safe driving of predefined groups of cars. Dynamic forming is an advanced problem beyond the current state of research.

2.4 Central and Distributed Approaches

Generally spoken, we can classify every network architecture in computer science as a centralized, a distributed or a hybrid pattern. If one or more clients are connected directly to a server, we talk about a centralized system. An everyday example is a website. Many different users are requesting pages. The server responds with the requested content. The server is the central node, while the users are clients. In decentralized systems, every node is client and server at the same time. Normally its function is predefined. The other way round, every node responds to requests from other nodes and sends them information. Figure 2.3 shows the setup of a central and a distributed system. While centralized systems usually have a star topology, distributed approaches may vary in the way nodes are connected to another. If any node has a direct connection to any other node, this is called a complete graph. The following traits are characteristic [2].

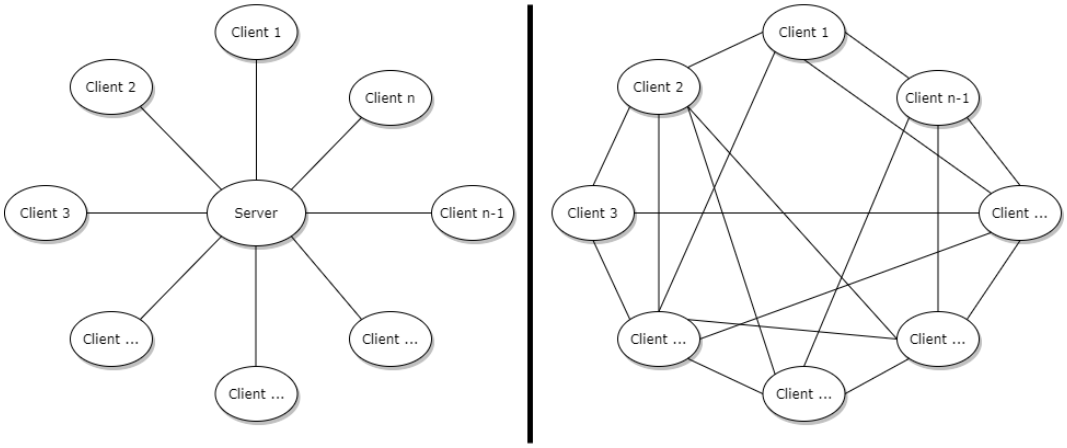

Figure 2.3: Visualization of a central (left) and a distributed system (right). The nodes represent clients and servers. The edges represent connections between two nodes.

- **Scalability:** Every node has a certain traffic limit able to handle in a specific time period. After passing a certain limit of clients there is no horizontal scaling possible because in a centralized system all requests are handled by one server. Of course, by adding another server or improving the performance of the old server, the system can improve. This is called vertical scaling. A distributed system can scale easily. There is no bottleneck, where every information has to pass.

- **Synchronization:** Since a centralized system has generally only a few servers, it is guaranteed that clients get the newest information. In the case of more than one server, it is necessary to synchronize the data on both servers. In distributed systems new information to a specific request may be noticed by a part of the system, while other parts are working with older information. It takes time to spread information over the whole network because there is no central instance. There have to be arrangements in order to remove old data.

- **Security:** A centralized system is easy to secure. By exchanging certificates, clients and server can prove their identity afterwards. In a distributed system no agent of the system has knowledge about all other parts so that malicious agents can join and infect the system. Distributed security mechanisms

do exist but they are highly complex in comparison to centralized security mechanisms.

- **Reliability:** In centralized approaches - if a single server fails, the whole system fails. In a distributed system, only some parts will crash. This is the strongest attribute of a distributed system. The idea of distributed work is, that if a part of the system fails, the rest is not affected and can still work properly.

In relation to our platooning system, we talk about a non-complete distributed system. A car represents a node. Every car has edges to cars inside its sensor range. This means it is possible for those cars to communicate directly. An objective is now to aggregate the decisions into an overall perspective. Forming a platoon means, that every car inside the platoon desires to stay in a group with the other cars.

2.5 The Multi-Armed Bandit Problem

The multi-armed bandit problem is a classic example of online decision-making in probability theory. It deals with the dilemma between exploration and exploitation. The classic scenario is derived from a casino: there are several slot machines each having a different but unknown winning probability. The problem is to develop a strategy to get the best long-term reward possible. Due to not having any prior knowledge, the player has to explore different actions in order to exploit the most-rewarding arm [29]. This may involve short-term sacrifices depending on the strategy. This problem is an analogy for a lot of common problems companies face every day, for example identifying the best advertisement to maximize business objectives or the click rate on a news feed. Therefore, this is not just a theoretical problem but has a great impact in practice. In a more scientific way, a multi-armed bandit has k arms. Since every arm corresponds to an action, we use this as a synonym. For each arm $a \in A$ there is exactly one corresponding probability distribution $\theta(a)$. At every time step t the player chooses an action $a_t \in A$ and reward r_t is sampled from $\theta(a)$. Because $\theta(a)$ is a probability distribution, the gained reward from the same action differs from sample to sample. The expected mean reward of arm a is defined as

$$Q(a) = \mathbb{E}[\theta(a)].$$

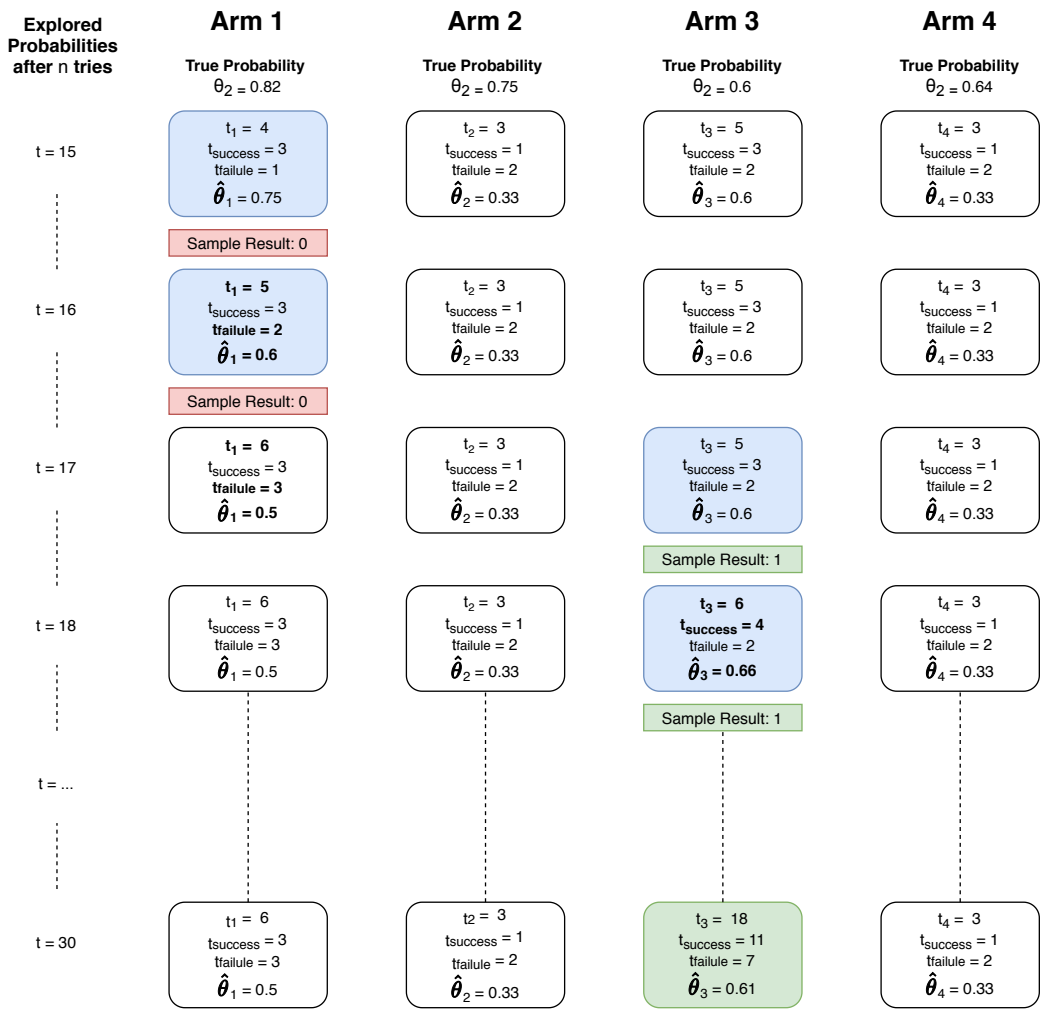

Figure 2.4: Example of a multi-armed bandit problem with a greedy solver. After $t = 15$ time steps, the first scenario occurs. $\hat{\theta}$ is the estimated probability at the specific time step. Blue marks the chosen action in each step. The box below the chosen action signifies whether it was a success or failure sample. Bold marks parameters, that have changed in the previous time step. The green box in the last step marks the decision.

$Q(a)$ is unknown. By picking more samples, we get a better estimation $\hat{Q}(a)$ of the mean reward. The objective is to maximize the cumulative rewards $\sum_{t=1}^{T} r_t$ over T time steps. This is the same as minimizing the total regret

$$L_T = \sum_{t=1}^{T} Q^* - r_t$$

where

$$Q^* = \max_{a \in A} Q(a),$$

is the best mean reward. The regret L_T is defined as the difference between the optimal reward Q^* and the reward of the chosen arm r_t summed over all time steps T. There are several modifications of the problem. The standard case is finding the best arm with an infinite number of tries. In other cases the amount of tries may be limited, so that the objective switches in order to find the best basis for decision making within a certain number of actions [29].

Figure 2.4 shows a typical multi-armed bandit scenario with a naive sample-based greedy approach. The true probabilities are not known to the player. After a certain amount of steps, the initial scenario in step $t = 15$ occurs. We directly start at $t = 15$ to show the conduct as well as the problems of a naive greedy approach. With this limited amount of samples, arm one seems to be the best solution. Due to the greedy approach, the option with the highest probability during the current time step is always chosen. In the following, due to two bad samples, the probability of arm one sinks to $\hat{Q}_1 = 0.5$. This leads to a switch. Afterwards arm three is chosen. Because the probability of arm three does not sink below $\hat{Q}_3 = 0.5$, we always chose arm three until step $t = 30$. With every added sample, the estimated probability of arm three approach closer to the truth. Even though arm three is the worst decision of all arms, the algorithm selects it as the best choice. This is a typical scenario, initial bad samples can lead to a bad decision. Hence, we need better algorithms for decision making.

The classic approach in decision making with no prior knowledge is performing multiple α-β tests. Afterwards statistical analyses identify the best variant. While this is possible for a smaller amount of choices (arms), for bandits with more than

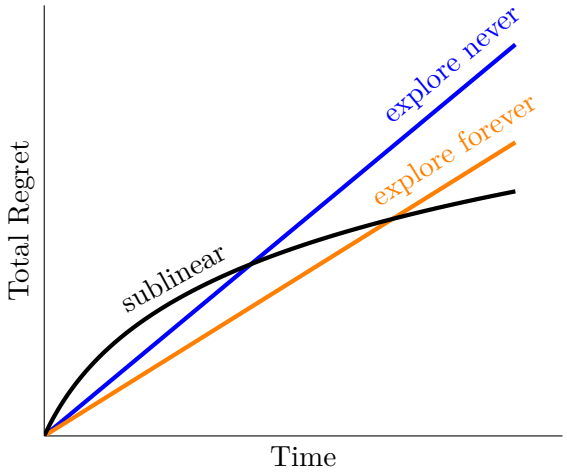

Figure 2.5: Schematic representation of regret variants. All classes of exploration-exploitation algorithms can be separated into this three groups.

ten arms, this time consuming method slows down rapidly. Since more variations require more tests in order to compare each variant with each other, this method terminates in $O(k^2)$, meaning the amount of calculations grows quadratically in regards to the number of arms. While in real world conditions one variant outperforms the others most of the time, comparing two non-optimal variations provides an unnecessary overhead. Another problem is, that this testing method does not give any knowledge, until the test has fully passed. To solve these problems, there are several algorithms from machine learning adopted to this task [29].

2.6 Machine Learning Algorithms

The multi-armed bandit problem is well researched. Especially in the field of machine learning, several algorithms have been developed to solve this problem in an accurate way. The following section is a summary of David Silver [8] and Russo et al. [26] with a focus on exploration and exploitation. In this section, we focus on three different approaches: Greedy Algorithms, Upper Confidence Bound (UCB) and Thompson Sampling. In common, all approaches have no prior knowledge. No algorithm knows anything about the properties of the distributions. To estimate the

probability in a random environment, the Monte-Carlo evaluation

$$\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{\tau=1}^{t} r_\tau \mathbb{1}(a_\tau = a), \tag{2.1}$$

is used. $\hat{Q}_t(a)$ is the estimated reward of action a at time step t from $N_t(a)$ samples. $\sum_{\tau=1}^{t} r_t$ is the total reward. We define the binary operator as $\mathbb{1}(a_t = a)$. The operator returns one, if the condition is fulfilled and zero otherwise. The binary operator guarantees that only reward r_t is summed, if action a was chosen in time step t. Before investigating those algorithms we notice a main observation shown in Figure 2.5. If an algorithm never explores, it always chooses the action with the supposedly highest value. Due to no exploration afterwards, such algorithms can get stuck at a suboptimal decision, which leads to a linearly increasing total regret. If an algorithm explores forever, at a certain time t this algorithm has found the best option. Because there is no knowledge whether this objective best solution really is the best solution, the algorithm keep on exploring. This also leads to a linear increase in regret. According to these facts, an optimal algorithm must find a good balance between exploring in the beginning, to find the optimal solution, while exploiting afterwards. In the following we introduce three widely spread algorithms for handling a multi-armed bandit problem, which were mentioned above.

- **ε-Greedy Algorithm:**

$$\hat{a}_t^* = \begin{cases} \arg\max_{a \in \mathcal{A}} \hat{Q}_t(a) & \text{with } \mathbb{P} \geq \varepsilon \\ \text{random} & \text{else} \end{cases} \tag{2.2}$$

  The ε-greedy algorithm selects with probability $\mathbb{P} \geq \varepsilon$ the action with the currently highest explored value. In rare cases (with probability $\mathbb{P} < \varepsilon$), a random action is explored. Initially all actions start with an optimistic initialization, thus meaning that all estimated rewards start with the maximum payout value. The selected action $\hat{a}_t^*$ is explored and its probability for the next turn is updated via Monte-Carlo equation (Equation 2.1). As mentioned above, a greedy approach may get stuck in a local maximum. For example by picking

unlucky samples for the supposed best option at first, this may exclude the actual best option forever. Because the algorithm always picks the action with the highest probability, the optimal action has no chance to be picked anymore.

By adding a random factor ε the algorithm finds the action with the best mean reward. Depending on ε this may take time. While a higher exploring chance avoids getting stuck, it also slows down the process of estimating the right mean reward for the best solution. With a fixed ε, the random factor leads to a linear regret after finding the best solution. In order to minimize this regret, it is possible to decrease ε over time. This method combines the advantages of forever explore and never explore, while reducing their disadvantages to a minimum. A sublinear regret is the consequence.

- **Upper Confidence Bounds (UCB):**

$$\hat{a}_t^* = \arg\max_{a \in \mathcal{A}} \hat{Q}_t(a) + U_t(a)$$

As opposed to a greedy algorithm, in UCB we are looking for the action $\hat{a}_t^*$ with maximum sum of probability $Q_t(a)$ and some uncertainty $U_t(a)$. If there is no prior knowledge about the distribution available, upper confidence bound algorithms are a good option. It is based on Hoeffding's Inequality [22]

$$\mathbb{P}[Q(a) > \hat{Q}_t(a) + U_t(a)] \leq e^{-2tU_t(a)^2}.$$

The probability $\mathbb{P}$ that the true mean $Q(a)$ is greater than the sum of the sample mean $\hat{Q}_t(a)$ and the upper confidence bound $U_t(a)$ at timestep t is smaller than or equal to a certain probability. In order to pick a bound with high chances of fulfilling this inequality, $e^{-2tU_t(a)^2}$ shall be small. The upper confidence bound is a function either of the amount of tries overall or the amount of tries of a certain action. This is dependent on how the UCB-part works.

One heuristic way is decreasing $U_t(a)$ over time. By picking a certain action more often, our certainty regarding that action gets stronger, thus reducing $U_t(a)$. $U_t(a)$ can be chosen differently. By setting $e^{-2tU_t(a)^2} = t^{-4}$ and rearranging,

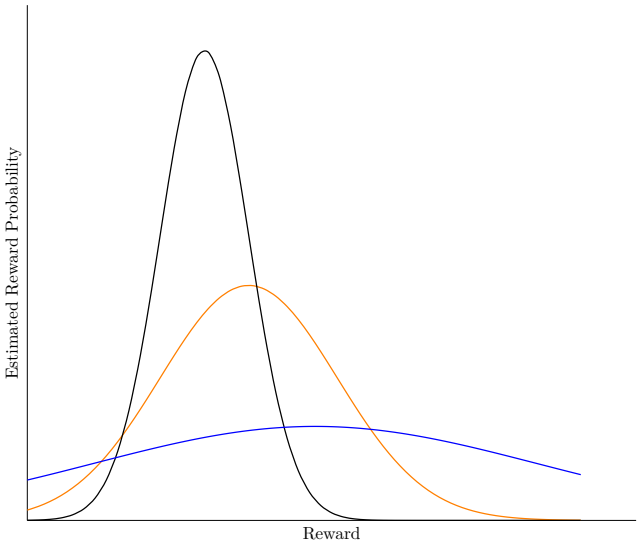

Figure 2.6: Schematic representation of a Bayesian confidence estimation. Blue marks the first estimation. The second estimation is represented by the orange curve. The last estimation is the black curve.

we get the *UCB 1* algorithm with

$$U_t(a) = \sqrt{\frac{2 \log t}{N_t(a)}}, .$$

While *UCB 1* is a general way of considering uncertainty, its biggest disadvantage is its generality. If there is some prior knowledge about the distribution, we modify the uncertainty $U_t(a)$. For example by knowing that it is a Gaussian distributed, we set $U_t(a)$ to an interval of twice the standard deviation. Figure 2.6 shows this process. By calculating the mean and standard deviation in each time step, we make the Gaussian curve more exact. First, due to a small amount of samples, we have a very flat and wide-spread curve (blue). A higher amount of samples shortens the deviation, so that the peak is getting higher (orange curve). If we have sampled a lot of actions, finally the uncertainty (deviation) is reduced to a minimum (black curve). With every sample

the uncertainty is reduced. This leads to

$$U_t(a) = 2\hat{\sigma},$$

where $\hat{\sigma}$ is the standard deviation of the estimated Gaussian curve.

- **Thompson Sampling:**

$$\hat{a}_t^* = \arg\max_{a \in \mathcal{A}} \tilde{Q}_t(a)$$

In Thompson Sampling we are looking for an action that has the highest chance to be optimal. At first, for each action we initialize a α-β-distribution based on prior knowledge. If there is no prior knowledge, a distribution with $\alpha = 1$ and $\beta = 1$ is recommended. At every time step, we take a sample $\tilde{Q}_t(a)$ of each action from the corresponding α-β-distribution. The action corresponding to the best sample is actually chosen for this time step and we are updating the corresponding distribution afterwards with the reward as follows:

$$\alpha_i \leftarrow \alpha_i + r_t \mathbb{1}[a_t = a_i]$$
$$\beta_i \leftarrow \beta_i + (1 - r_t)\mathbb{1}[a_t = a_i]$$

With every step, the α-β-distribution is getting more accurate, leading to a probability matching like approach of estimation.

2.7 Platooning Extension for Veins (PLEXE)

Segata et al. [27] created PLEXE[1], which is a comprehensive platooning simulation based on a complex substructure. The core is the Simulation of Urban Mobility software (SUMO), a road traffic simulation for the microscopic representation of any complex scenario [7]. The tool combination PLEXE and SUMO is our choice for simulating platooning behavior. SUMO is an open source project from the German Research Center for Air and Space Travel. The currently available software gets frequently updates and bug fixes. A community uses SUMO, thus resulting in well-documented projects and tutorials. Using this software combines a detailed and well-working road traffic simulation with the advantages of an open source project.

[1] `http://plexe.car2x.org/`

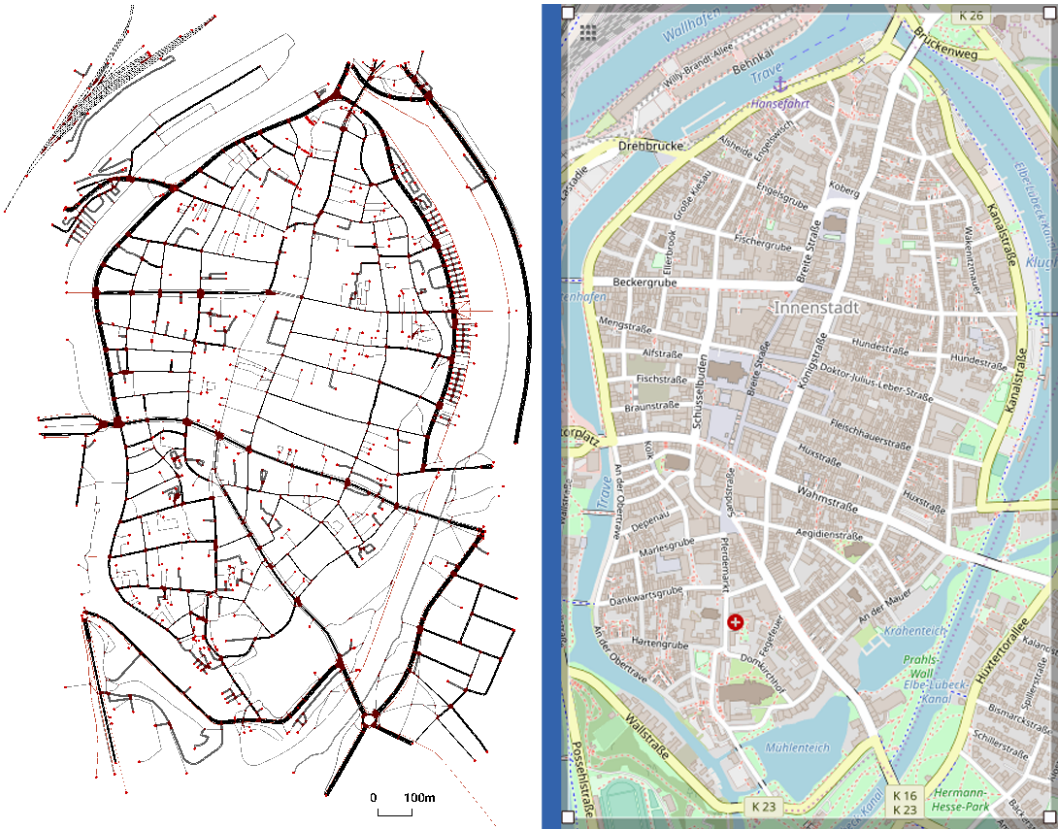

Figure 2.7: This picture shows a map view of the historic district of Lübeck city. On the right side is the map from Open Street Map. On the left side one can see the same map exported from Open Street Maps and converted to a network graph. The small red nodes all over the map are crossings. The black lines in between two red nodes are possible roads to drive on. End points, such as parking lots or gateways are red nodes as well.

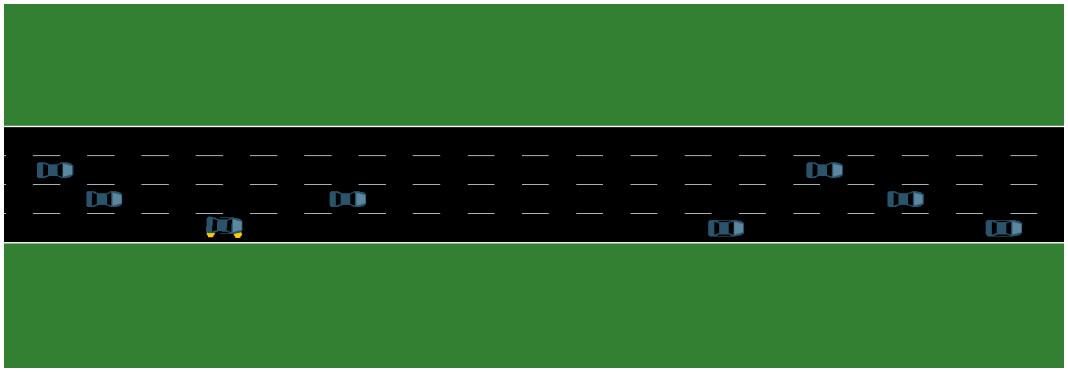

Figure 2.8: Simple example of traffic on a highway. Vehicles are spawning every second. No V2X communication is established. With radar sensors, cars are able to recognize cars in front of them and their speed in relation to their own. Cars with a higher speed overtake slower cars if possible. Therefore they check the overtaking lane. If it is empty, a lane change takes place.

No inner vehicle communication (V2X) is supported within SUMO. However, V2X is required as the basis for platooning. The framework Vehicles in Network Simulation (VEINS)[2] is based on SUMO and supports the requested V2X communication. On top of that, there is PLEXE as a platooning framework extension to VEINS. The extension provides several methods to organize cars in groups, such as control algorithms for platooning, line changing behavior and rearrangement methods for platoons. It is also possible to get a lot of car related data such as radar, speed and position. This makes it easy to develop own algorithms and extend the framework. SUMO, VEINS and PLEXE are written in C++ language. For PLEXE there is also a Python API available. That is a wrapper to address C++ code in Python.

In order to arrange a street network, there are several tools included. You can choose to create networks on your own via the built-in tool NETEDIT [7]. This software provides a graphical user interface for network graphs. An edge implies a road, while nodes symbolize crossings. The road setup allows for specific detailed information, such as the number of lanes, speed limitations or lanes with special

[2] `http://veins.car2x.org/`

tasks (for example turn lanes). To simulate traffic in realistic scenarios, NEDEDIT also allows to import existing road networks from specific formatted XML files, for example open street maps format, [3] as the example in figure 2.7 shows.

There are two ways to simulate traffic - either offline via an XML file or on-line via the Traffic Control Interface (TraCI) [7]. Offline it is possible to specify routes and create vehicles driving along a specific route. Vehicles need informations about their type of car, starting position and at which time step the car appears at the starting position. Vehicles can also be generated in a stream. This means, within a certain time distance a car repeatedly spawns at a starting position and follows the route. A stream can be limited to a certain amount of cars or time distance related. The online creation via TraCI allows the same possibilities with the exception of streams. In exchange there is a lot of detailed behaviors, which can be set up with TraCI in the beginning or during the simulation. TraCI is accessed directly in program code. Figure 2.8 shows a simple example of traffic on a highway.

[3] `https://www.openstreetmap.org/`

3 Methods

Chapter 2 provides a basic understanding about platooning, the multi-armed bandit problem and related knowledge. In the following, we adapt platooning as a multi-armed bandit problem as mentioned in section 1.2. Our main objective is to increase the sum over the overall efficiency of all cars by allowing a dynamic reconfiguration of platoons at runtime. Static platoon formation (initially at the beginning of an experiment) will serve as the baseline for comparison. In order to reach that objective, we use methods from machine learning that were developed to solve the problem of multi-armed bandits [12]. It is the problem of choosing one of several actions. Each action refers to a reward. We want to maximize the expected gain and the profit of each action is only partially known. Key is to balance the exploration-exploitation trade-off.

We evaluate existing scenarios in section 3.1. In section 3.2 we develop a suitable model for our further work on the basis of this analysis. This includes the set up of roads (for example exits, amount of lanes, length) as well as the set up of vehicles as state machines. With the given aspects of this section, we have now identified a scenario fitting for platoon formation. We have developed a system, which is suitable for platoon forming algorithms. Before we go on with adapting algorithm from the machine learning sphere to our problem, we take a look on the realization in the simulation software PLEXE-SUMO. We separate section 3.3 in a general overview of the framework extension, the vehicle creating process and the road arrangement. Afterwards we adapt multi-armed bandit solving algorithms in section 3.4. We implement the algorithms ε-greedy, UCB 1, Bayes UCB and Thompson-Sampling. As a baseline, we implement the approach of Heinovski and Dressler [14]. Finally we are summarizing the expected behavior and problems in section 3.5.

3.1 Analysis of Distributed Platoon Forming Approaches

We want to gain an overview of aspects and currently not dealed problems in published scientific work related to our topic. We focus on a deeper analysis of previous platoon forming approaches. We start by contrasting centralized and distributed solutions. The concrete comparison is only possible to a limited extent because in some cases completely different approaches were chosen. The platoon formation works fundamentally different. We can compare the final results, but can not compare the routines leading to these results. Instead, we focus on sub-aspects and evaluate them one by one. We want to get an idea of the importance of each aspect regarding to our problem. Each of these solutions describe a specific setup. In each setting, there are restrictions. These have been introduced to reduce complexity or because certain behaviors are negligible for the scenario being studied. At the end of this section we will focus on the used communication mechanisms. Here the focus is on finding suitable methods for our setup.

3.1.1 Advantages and Disadvantages of Distributed Approaches

Heinovski and Dressler [14] provided a centralized and distributed approach for platoon forming. This is a good starting point for comparison because there is a big similarity in the used algorithms. Both algorithms are based on the same model. Each car contains a set of parameters,

$$\{\text{id, des, pos}\}, \tag{3.1}$$

where *id* is the identifier, *des* the desired speed and *pos* the current position of the car. This forms an optimization problem with the aim of

$$\forall_i : \text{minimize } f_i(x), \forall x \in \Omega_i, \tag{3.2}$$

with the function

$$f_i(x) = \alpha d_s(x, i) + \beta d_p(x, i), \tag{3.3}$$

where Ω_i is the neighborhood of car i and hence x represents a neighbor. α and β are weighting factors will the constraint: $\alpha + \beta = 1$. The functions for the speed

deviation $d_s(x,i)$ and position deviation $d_p(x,i)$ are defined as

$$d_s(x,i) = \|\text{des}_i - \text{des}_x\|$$

$$d_p(x,i) = \begin{cases} \|\text{pos}_i - \text{pos}_x\|, & \text{if } \text{pos}_x > \text{pos}_i \\ \infty & \text{if } \text{pos}_x \leq \text{pos}_i \end{cases}$$

with respect to

$$d_s(x,i) \leq p \times des_i, p \in [0,1],$$

$$d_p(x,i) \leq r,$$

$$\alpha + \beta = 1.$$

The constraint $d_p(x,i) \leq r$ with range r is the description of $\forall x \in \Omega_i$, for every car x in range r to car i.

Heinovski choses a greedy approach. This approach does not get the optimum in most of the simulations. By computational reasons, it is a heuristics, that outruns any time consuming linear solver. As the complexity of neighborhood relations to be calculated is in $O(n^2)$, fast algorithms are important, to also make a platoon formation possible in situations with a huge amount of neighbors [14]. A list is generated with values from the functions of all neighborhoods and sorted by value. From these list they choose the lowest value. The two affected cars perform a joining maneuver. All other list entries calculated on the basis of one of these joining cars are deleted. This procedure is repeated until no more pairs of vehicles are found. The difference between the central and distributed approach is the content of the list. In the central case, the list is constructed as described above. In the distributed approach, each vehicle builds its own list. For this it relies on information that a vehicle receives via its sensors. The quality of decision depends on the quality of the sensor data as well as evaluation time.

Because of the similarity, both approaches can be compared well. Heinovski's results are separated into the following categories:

- **Number of Platoon Candidates:** The centralized approach finds fewer candidates per vehicle than the distributed one. This is due to the omniscient

observer who sorts out cars in a joining maneuver as not selectable. In the distributed approach, an agent does not have that knowledge. It gets knowledge sample by sample. This results in more overhead, especially when considering options that will not work anyway.

- **Number of Joining Maneuver:** A bigger number of candidates highly increases the possibility of aborting a joining maneuver. An abort happens if a car wants to join another car, which is currently in a joining process. This problem is affected by the amount of platoon candidates found, so that the distributed approach has a higher number of joining maneuvers (and thus more aborted joining maneuvers) as well.

- **Platoon Size:** The restriction *Cars in a platoon are no longer able to join another platoon* [14] influences this category. The centralized approach tries to match as many cars as possible. Because there are not that many not-yet-platooned cars left after the first wave of joining maneuvers, this leads to a small platoon size. The distributed approach leads to a higher number of joining maneuver aborts. Cars aborting such a maneuver stays in a single car state, so the pool of not-yet-platooned cars remains on a higher level. Only single driving vehicles can join a platoon. It is easier to find another member for formed platoons, because there is a higher number of not-yet-platooned cars.

- **Desired Speed Deviation:** Considering the restriction *The leading car allocates the platoon speed*, [14] in smaller platoons only a few cars have bigger deviations from their desired speed, while it is the other way round in bigger platoons.

- **Travel Time:** To drive in a platoon, an agent has to make a desired speed compromise. Each platoon member wants to drive a desired speed. Because the platoon leader sets the driven speed, every car has to accept deviations from its own desired speed. Another aspect are joining maneuvers. They can take time, especially if a car has to slow down or even abort the joining process due to outer circumstances. The travel time is quite similar in both approaches. In the distributed approach bigger platoons lead to a bigger increase of travel time. The platoons in the centralized approach are formed faster. If driving

in a platoon, most of the time a vehicle can not drive at its desired speed. This leads to a loose of time, when reaching its destination. Benefits from platoon formation like street usage efficiency and avoiding traffic jams are not considered here.

- **Fuel Consumption:** Again, the fuel reduction is similar. While bigger platoons are more efficient at saving fuel, the faster the platoon forms, the earlier fuel saving effects occur.

There are a lot of other approaches in this field of study. We focus on the approaches of Larson et al. [18], Liang et al. [19] and Caballeros Morales et al. [6], because all of these solve platoon formation as an online problem with a distributed approach.

The algorithm development of Heinovski and Dressler [14] is based on Caballeros Morales et al. [6]. They presented the so-called *Adaptable Mobility-Aware Clustering Algorithm based on Destination positions* (AMACAD). This is a mixed algorithm situated between centralized and distributed due to each agent having knowledge about its environment. This knowledge is strictly local. The agent does not have a general overview about the overall situation. As the name suggests, this algorithm takes the destination into account. The clustering function is based on

$$\Delta L_{v,z} = \sqrt{((Lx_v - Lx_z)^2 + (Ly_v - Ly_z)^2)},$$

$$\Delta S_{v,z} = |\text{Speed}_v - \text{Speed}_z|,$$

and

$$\Delta RD_{v,z} = \sqrt{((RD_v - RD_z)^2 + (RD_v - RD_z)^2)},$$

where $\Delta L_{v,z}$ is the distance between two cars v and z, $\Delta S_{v,z}$ the speed difference and $\Delta RD_{v,z}$ the distance between the destination of cars v and z. This is summed and weighted by

$$F_{v,z} = w_1 \Delta L_{v,z} + w_2 \Delta S_{v,z} + w_3 \Delta RD_{v,z},$$

with constrains

$$w_1, w_2, w_3 \in [0, 1],$$

$$w_1 + w_2 + w_3 = 1.$$

The agent with the lowest sum of values over all agents in the neighborhood is chosen as cluster head. This is done via several base stations, calculating the sum for a subset of agents. Base stations are a distributed network of stations, that register vehicles in their range. Cluster heads lead a platoon for a not determined period of time. Each member, as well as the cluster head have knowledge about other cluster heads in the predefined neighborhood. In the case there is a better fitting cluster head, the members can choose another head known to them. The objective is focusing on stability. Therefore evaluation parameters, such as cluster head lifetime, membership lifetime and reaffiliation rate are chosen. The lower reaffiliation is, the longer lifetimes are thus leading to a more stable system. The main understanding of the results from Caballeros Morales et al. [6] is that AMACAD is much better in comparison to other algorithms ignoring the destination of an agent. But the algorithm is highly dependent of the chosen weighting parameters. Heinovski and Dressler [14] avoided this aspect by using testing setups, where every agent has the same destination. This leads to the conclusion, that especially in multi-agent scenarios with a lot of different destinations it is important to consider the destination as a necessary factor for forming a platoon efficiently.

Liang et al. [19] is the succeeding work of Larson et al. [18]. It is an overall study about the fuel consumption benefits in real world situations and the German Autobahn network is used. Although they have used a distributed approach, there is one major difference to Heinovski and Dressler [14]. The distributed stations are no longer cars - therefore the road system is modeled as a graph. Every node collects data of cars. Agents, driving on edges, send their data to the next node incoming in their direction. The aim of every node is to decide whether to form a platoon. New considered aspect is a time limit for arriving at a destination. The speed is adjusted by a relation of remaining distance and remaining time. A speed limit exists as well. Therefore in some cases, considering truck platooning as the crucial part of introducing platoons to the real world, there is no possibility for an agent to join a platoon. Another speed-dependent aspect of joining a platoon is the fuel saving rate in comparison to the acceleration phase in order to get to the platoon. In general, it does not make sense for a platoon to speed up letting a vehicle in front join them, since a speed up comes along with a higher need for fuel. As studies mention an average fuel saving of about 10%, the benefits of a single car must be

superior to tolerate disadvantages of platooning for every car (for example speed adaption). Moreover, they mention the limited allowed maximal driving time in one session (e.g., 4.5 hours for trucks in Europe) to be considered as an important part for adjusting platooning approaches into the real world. To summarize, Liang et al. [19] and Larson et al. [18] had the intention to present an approach working well under real world conditions. Therefore they considered more aspects than the other approaches.

All in all, this leads to a huge amount of aspects, we may consider. We must identify the relevant factors for our scenario. This must take the balance between detailed aspects for a scenario close to real world circumstances and minimizing aspects for focusing on our goals.

3.1.2 Communication Mechanisms

For platooning, a car needs knowledge about several details of other vehicles. This can be the position of neighbors in order to form a platoon with one of them or information about a car's own platoon, like information about the leader or the position inside the platoon. For passing on information to other vehicles, a V2V communication system is crucial [33]. In his master's thesis, Hobert [15] presents communication methods for Adaptive Cruise Control (ACC) and Cooperative Adaptive Cruise Control (CACC). At the time of his work platooning systems were only able to automate longitudinal movement. Lateral automation, for example needed for overtaking maneuvers, could not be handled. Just half a year later, Bergenhem et al. [1] published an overview of five intelligent transportation system (ITS) projects. It shows, that with SATRE [25], PATH [28] and ENERGY ITS [31] there are at least three projects, which allow lateral automation.

The simulation PLEXE offers an auto feed method, simulating inter vehicle communication. However, this is no V2V communication for a real environment [27]. Since the main objective of this work is the dynamic grouping and rearranging of platoons, we can rely on that auto feed and do not have to struggle with problems of inter vehicle communication. For our scenarios, we assume there is a stable and reliable V2V communication used.

3.2 Model Setup

We define some keywords for better understanding of the model overview. A *neighbor* of a vehicle is a car, that is in sensor range of the vehicle. The *platoon leader* is the first member of a platoon, that drives with ACC and is the initiator of driving maneuvers (section 2.3).

The happiness is a vehicle to vehicle relation, that indicates the similarity between both cars. Each car can calculate its happiness to another car. The *current happiness* of a vehicle is the relation of the car to its current platoon leader. If the car is in no platoon, the *current happiness* is the happiness to itself. The happiness is separated into components, each weighted by a factor. Each component returns a value in the interval $[0, 1]$. The weighting factors are normalized, so that

$$\sum_{i=0}^{c} \alpha_i = 1,$$

where α_i is the weighting factor of the i-th component and c is the amount of components.

From all the aspects mentioned in section 3.1, it is our objective now to figure out how to set up the model. This includes the aspects choosing important features for platoon comparison, how to measure the success rate of formed platoons and restrictions for a better scenario handling. On the one hand, restrictions are made to simplify the setup. On the other hand they help keep focusing on specific aspects.

Section 3.2.1 gives a basic understanding about the scenario and processes, such as routing of cars and platoon candidate search. We implement each car controller as a state machine. Section 3.2.2 explains the defined states and their connections. Crucial for qualifying our scenarios is a success measurement. From the information of section 3.1 we identify the appropriate parameters for our happiness measurement in section 3.2.3. Homogeneous platoons have high happiness, while heterogeneous ones stay low and disrupt themselves. The happiness is an indicator of how many compromises a vehicle takes to stay in a platoon.

3.2.1 Model Overview

In section 3.1 we analyzed the approach of [14] as a static approach. This means, that cars stay in a platoon until the platoon leaves the simulation or the simulation ends. Platoon changing is not allowed. Creating a more dynamic platooning environment is our main overall objective. The following idea is our ideal solution. From the start, we investigate the behavior on a highway with two or more lanes. There are different entry / exit stages along the highway. In between, there are distances of about six kilometers to travel. This serves well for situations with a lot of platoon formation and disruption (e.g. cars enter / leave the highway, accompanied by cars leaving platoons and disappearing from the scenario - other cars entering the highway and joining a platoon soon), as well as for stable situations on driving in between these highway entries, where cars can only get in sensor range slowly. Cars randomly appear at the starting point or on the entries of the highway. The desired speeds and destinations differ between cars. Every car detects neighbors. In an ideal situation, the car chooses a candidate vehicle from its neighborhood in order to form a platoon. All candidates are vehicles in the car's sensor range, who are leader of a platoon. After choosing a candidate, the single car speeds up for a certain time to come in range of the candidate. When getting in touch, a car checks crucial conditions (e.g. if the neighbor leaves the highway immediately, a candidate to form a platoon is of no use). If this check is passed, a car calculates its happiness relation and compares it to the happiness of potential candidates. Depending on the implemented algorithm, a car chooses a fitting partner / group on the basis of this happiness. Being in a platoon does not break this routine. A member of a platoon continuously checks its current happiness of its platoon. If there is a better matching platoon with a happiness increase higher than a predefined threshold in comparison to the current match, the car leaves its platoon and changes to the better one. If the happiness decreases to a low level and no other platoon is available, a car decides leave the platoon and go on its own again.

With this setup idea, we create a dynamic scenario with joining and leaving maneuvers. The detailed work on parameters, features and algorithms is now essential for the behavior of this system.

3.2.2 Vehicle States

In order to implement the model, we introduce several vehicle states. In a certain state, a car does specific actions and is able to change into another state. This state machine helps to structure the behavior in a more comprehensible way. The state machine is shown in Figure 3.1.

When created, a vehicle always starts in the state *spawning*. In this state the vehicle has some time to adjust to its parameters. This is either a start period, if spawned on the highway directly, or the entering process on a highway entry. Afterwards it switches to state *singe car*. In state *single car*, a vehicle drives regularly on its own. The car is looking for possible matching candidates in order to form a platoon. When the specific algorithm returns a candidate, the vehicle switches into *prepare joining*. Here, the vehicle tries to fulfill the prerequisites for platooning. This means, that it changes lanes to be on the candidates lane. Afterwards it is closing up. If the car gets in a certain range to the candidate, it switches to state *joining process*. In this state, the platoon is set up, afterwards the state immediately changes to *platoon*. While in the state *prepare joining*, it may occur that forming a platoon with the candidate is not feasible anymore. Either it is out of range or the highway in between both vehicles is too crowded. In these cases, the car switches into *no platooning* state.

Going on with *platoon* state, there are several ways to leave / re-enter a platoon. The candidate matching algorithm is looking for better matches. There is a platoon switching threshold. If the happiness difference between the current platoon and a new vehicle is above this threshold, the algorithm delivers the new vehicle as a candidate and the car switches to *prepare leaving*. If the happiness of the car to its current platoon sinks below the predefined minimum value or the end of its route is close, the car switches into *prepare leaving*, too. The minimum value is equal with the happiness a car has while driving alone. If a platoon leader detects a candidate, it does not try to leave the platoon. Instead it starts a merging progress in the state *merging*. During the merging process, the platoon leader does the same as described above for a single vehicle in the state *prepare joining*. Furthermore, its platoon members follow. For the state machine, it does not matter whether the

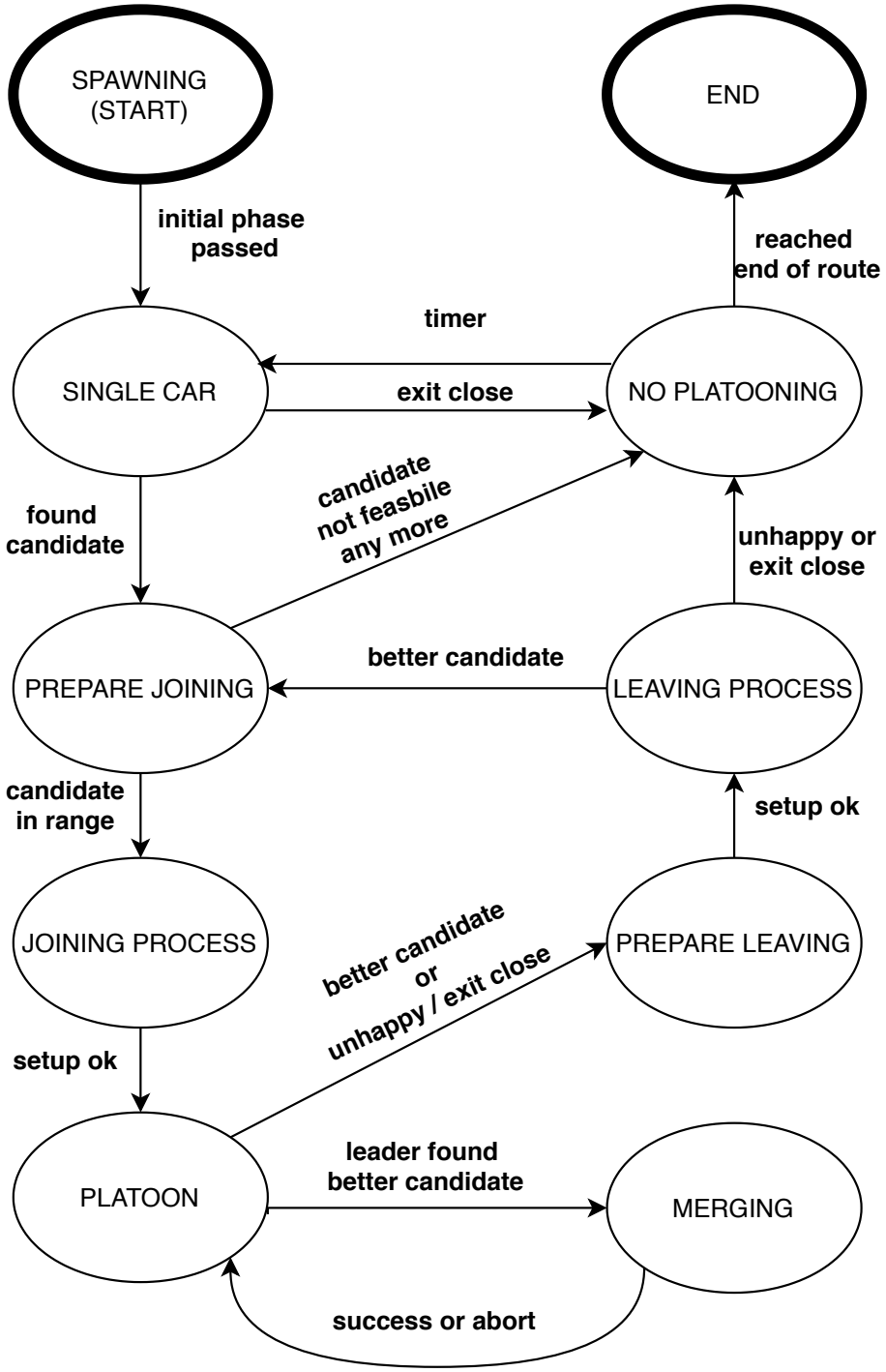

Figure 3.1: State chart of vehicle implementation. The circles mark states. A vehicle can change into another state, if both states are connected with an arrow. Therefore the vehicle has to fulfill the prerequisite next to the arrow. Start and end state are marked in bold.

merging process was successful. In both cases, the state switches back to *platoon*. In one case, the referring platoon is now merged into another one and has stopped existing. In the other case the platoon exists as before.

When trying to leave a platoon, a car can not leave instantly, while being in platoon formation. The minimal safety gap between platooning and non-platooning cars is the problem. A direct leave would result in an emergency braking of succeeding cars, thus leading to a platoon disruption. Therefore, in the state *prepare leaving* the leaving car switches lanes in order to leave the formation. When matching the pre-requisites, the state switches to *leaving process*. *prepare leaving* and *leaving process* adopt the exact opposite of *prepare joining* and *joining process*. When in the state *leaving process*, the car either switches to *no platooning*, if the car wants to leave the highway soon or was unhappy, or it switches to *prepare joining*, if the reason for leaving was a better matching candidate. If a car leaves due to unhappiness, it should not match with the same platoon again. Therefore *no platooning* is a state of waiting, in which a car can not join a platoon any longer. Only after a certain time it switches to *single car* again, thus leading to new platooning options. A car leaving the highway does not switch from this state any more. A car removed from the simulation is always in the state *no platooning*.

3.2.3 Success Measurement

We are looking for properties, that are strongly relevant for the platooning behavior of each car. It is important to group most homogeneous vehicles in order to form a sustainable platoon. The ideal setup is a group of vehicles in which are all sharing the same desired speed and destination. Within this group every vehicle is able to reach its destination with a minimum of time. This means that the **desired speed** and the **desired destination** are important features to form a platoon. One factor of main importance is the **current platoon size**. In platoons with a smaller amount of members the speed of the platoon may change in a non-favorable way caused by joining processes in the near future. A platoon with a big amount of members close to the maximum reduces the possibility of major speed changes of the platoon. We are loosing uncertainty about the behavior. Finally there is the **current distance between a candidate and a platoon**. In general it is easier to close a smaller gap

between a car and a platoon. It is a minor feature because it is only important for joining processes and has no influence afterwards. We introduce a weighing factor for each aspect in order to fit the importance of these features to the scenarios. The weighing factors vary from simulation to simulation.

The main profit of platooning is by all means reducing fuel consumption. Davila et al. [9] show, that the distance between each vehicle does have a great impact on slipstream fuel savings. Besides the happiness of each vehicle needs to be as high as possible. A vehicle does not participate in a platoon at any price. Platooning means to make a compromise between fuel saving on the one side and speed reduction on the other, thus arriving later at the destination. Since we established a happiness function to bring the desired needs of a vehicle into an abstract form, we should compare the overall happiness as an indicator of how well the decision making algorithm works in different simulation settings. As a counterpart, we can observe the fuel consumption related to the current happiness. Since happiness is strongly dependent on the chosen weighing factors, a bad choice of that parameter can weaken the relation between happiness and fuel consumption. A small weighting factor for the size of a platoon results in a low happiness, if the average platoon size is high. Therefore we observe different parameters. The average platoon size is a strong indicator for fuel savings. Because only the first vehicle in a platoon can not profit from slipstream savings, obviously the fewer platoon leaders and single driving cars we have, the bigger are the saving rates. Desired speed in relation to the real driving speed is a good indicator for happiness. Our objective is to minimize the difference between desired and actual driving speed. Speed peaks are representing long distances between a joining car and a platoon, because a joining car speeds up in order to get connected to the platoon. Nevertheless it does not say a lot about fuel savings except that in general a lower speed leads to a lower fuel consumption. The desired speed in combination with the desired destination is an indicator of a homogeneous platoon.

3.3 PLEXE

The PLEXE API is available in C++ or Python, where the Python API is just a wrapper for accessing the C++ based code from PLEXE in Python. Both APIs are

offering the same pool of methods. While the C++ API offers an analyzing framework, this is not included in the Python API. In exchange the Python version has methods for simulating inter-vehicle communication, meaning the user can choose whether to take care of the communication process or just rely on the included auto feed system. Since communication is not a major part of this work, the auto feed method offers great benefits as a starting point. As Python has great possibilities for data analysis, an analysis framework is simple to implement. We use the Python API, as this version offers a high user friendliness. The PLEXE API provides a pool of V2X and platooning mechanisms [27]. Meanwhile most of these methods are only working in an already formed platoon. The joining and leaving processes are not implemented yet. The framework needs to be extended for these mechanisms. Furthermore PLEXE allows to create road networks of nearly any degree of complexity. For testing algorithms, we have to choose one network fitting to our purpose.

3.3.1 Framework Extension

The built-in vehicle data structure from PLEXE serves well for getting data belonging to each car. This includes acceleration, position, time, vehicle length and radar sensors. Nevertheless it lacks in information we need for platooning. Therefore we need to expand the existing data structure for platoon-related information, such as leader, members and the position inside a platoon. The program code written for this thesis is directly accessible via QR code in figure 3.2.

Figure 3.3 shows an UML diagram of the structure of PLEXE and the planned extension. As mentioned in the previous chapter 2.7, PLEXE is based on the detailed simulation tool SUMO. In recent versions PLEXE has been migrated into SUMO, therefore it is not mentioned as a separate package in the diagram. The two ways possible for communication with SUMO are Plexe Python and TraCI. TraCI can be separated into multiple subclasses. Mainly we use the subclass vehicle for getting and setting information of cars while running the simulation. Plexe Python instead offers a pool of basic methods of V2V communication. This creates great efforts in managing the platooning process. To avoid grabbing data from several different packages, we are not addressing the TraCI vehicle class directly, therefore extending it with the class *VehicleDataStructure*. For each vehicle an instance of

Figure 3.2: QR code for accessing the program code of this thesis. The Git Hub repository is either accessible via the QR code or directly via URL: `https://github.com/JSchwarzat/Platooning_Masters_Thesis`

that class is generated. It stores all data of a vehicle. On top of that, every information from TraCI is requested via *VehicleDataStrucure*. Note, that this class does not fully support all TraCI vehicle commands. It only supports methods, which are necessary for the current setup. Furthermore it is easily extendable for other TraCI methods, since it is just referencing to them.

The main class manages the administration of vehicles. It runs the simulation, creates the requested cars and coordinates the simulation process for every car. Helpful algorithms supporting the management are separated thematically into three subclasses. The class *createUtils* has a pool of methods for creating stand-alone vehicles as well as platoon vehicles and set them up correctly. *platoonUtils* serves with tools for managing the platoon forming processes. It offers the two most important functions for adding and removing a platoon member. To manage that process, there are several auxiliary functions included, described below. The class algorithms contains methods for choosing the best neighbor in order to form a platoon. Several different algorithms are implemented in this class. Since TraCI offers relevant functions for processing the simulation, we access a minor subset of TraCI methods directly from the main class. This includes the *addCar* method in the TraCI vehicle class. We need to create a car first, before accessing it with a newly

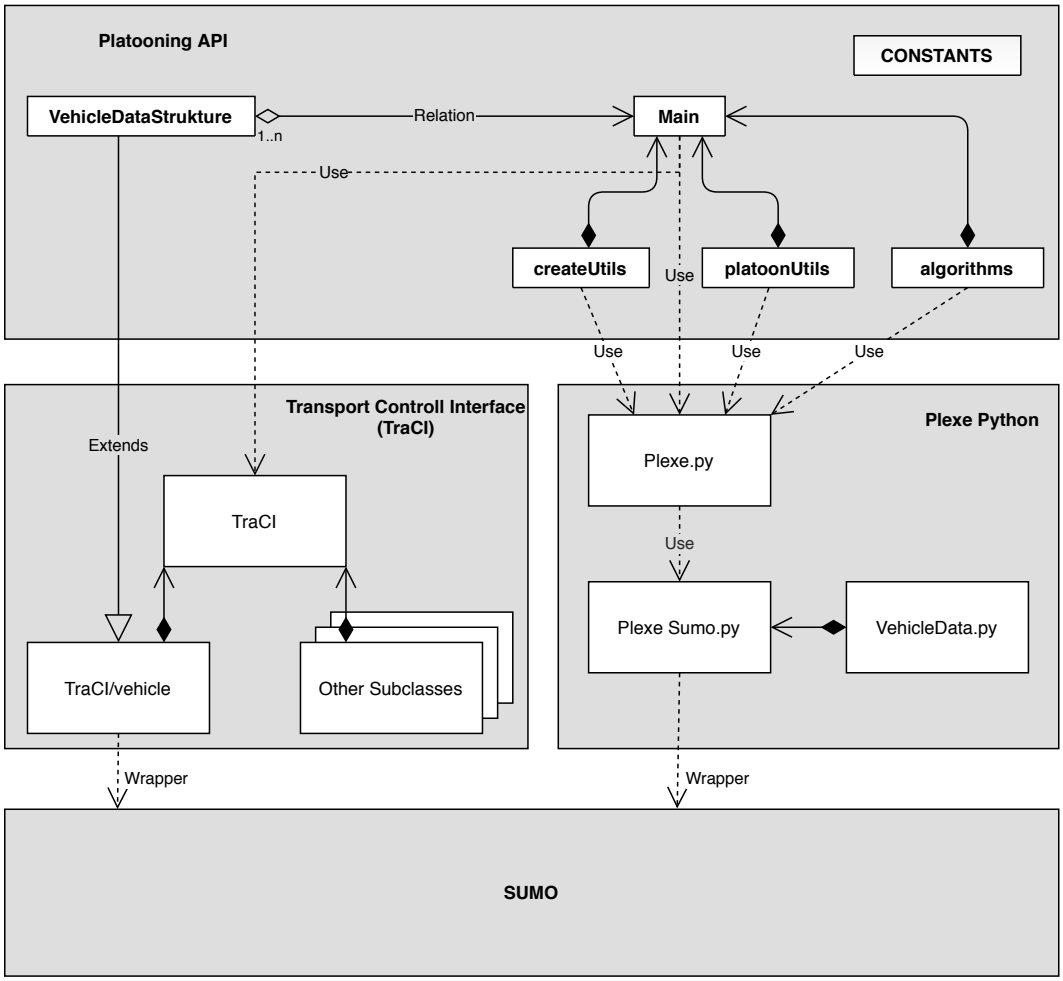

Figure 3.3: Overview of the structure of the implemented Platoon API. The grey boxes are separate packages. SUMO, TraCI and Plexe Python are open source. The package Platooning API is implemented during this work. The white boxes symbolize classes. On several arrows, there are descriptions of the relation between two classes. The arrow with a black rhombus is a composition, meaning the class next to the rhombus is a strong dependent part of the other class. The arrow with the gray rhombus is an aggregation, meaning that the Main class has one or more instances of *VehicleDataStructure*.

created instance of *VehicleDataStructure*. In general we can say, every getting or setting information process about a vehicle in TraCI is managed via instances of *VehicleDataStructure*, while not car specific information getting and setting is managed via the Main directly. The *Main* class as well as all mentioned subclasses are using basic methods provided by Plexe Python. The class *VehicleData* inside Plexe Python offers some data about vehicles. All data stored in this class is accessible via TraCI as well. Because of that, this *VehicleData* class of Plexe Python is not used. We use Plexe Python only for accessing basic platooning functions.

Furthermore we require methods for the platoon-forming process. In the following we describe the mentioned classes and the most important subset of methods.

platoonUtils

This class contains methods, which are necessary for the technical function of platoon forming, merging and leaving. Furthermore it delivers auxiliary functions, that the platoon related methods need for working properly.

- **init platoon leader:** When a platoon is newly formed, the leader must be set up. Besides vehicle information changing, the controller is switching to ACC and the leader is allowed to perform a lane change.
- **add platoon member:** This method adds a member to a platoon leader. The member needs to know its platoon leader as well as the car in front. Since the speed of the platoon is the median from all vehicles inside, it has to be calculated again after each joined vehicle. The controller is switching to CACC. Because the leader decides about platoon driving maneuvers, lane changing on its own is not allowed anymore.
- **prepare joining:** This routine does all the work, before a car is able to join a platoon. When a candidate is detected, the vehicle changes lanes to the desired platoon leader's one and speeds up in order to reach its desired platoon. When in platoon range, the platoon formation process takes place.
- **prepare for remove:** Platoons are driving without any safety gap between vehicles inside the platoon, but in care of the safety gap between cars from

another platoon. If a member leaves the platoon immediately, safety margins are neglected. Therefore this method prepares the leaving process. Depending on the reason for leaving, the leaving car or the rest of the platoon changes lanes as soon as possible.

- **remove platoon member:** This method removes a platoon member. If the leaver is a normal member, its information about the platoon is being reset. Furthermore its controller changes to a normal driving controller. The positions of the successors have to be reorganized, as well as the CACC algorithm. After the leaving process, the predecessor directly feeds the former successor with platoon-relevant data. If the leaving car is the platoon leader, his successor is the new leader. In that case, the successors controller changes from CACC to ACC. For a properly working CACC algorithm, the leader information must be changed in all platoon member vehicles. In case of only two vehicles inside a platoon, the platoon breaks. Therefore both cars are reset to single driving cars.

- **handling merging process:** This method is similar to *prepare joining*. Instead of closing up to the desired platoon alone, the same action is now done as a platoon coordinated action. When in range of the desired platoon, the succeeding platoon merges into the preceding one.

createUtils

This class delivers two methods: a single car and a platoon creation process. In both cases vehicles are created, depending on the given parameters. Parameters are the route, starting lane, and starting speed. All parameters are chosen randomly, if there is no parameter given. A Gaussian distribution shown in section 3.3.2 chooses the speed. The starting lane is chosen in relation to this speed. The route is randomly chosen as well, as long as all routes have the same probability. A route has a starting and an ending point. Each car can leave the highway at all existing exits in front of it. This results in $n + 1$ possible routes, where n is the amount of exits along the highway (n exits and going on). Overall there are $\sum(n + 1)!$ routing options.

algorithms

This class is the core of the platoon choosing process.

- **find candidates:** This method gets a list of all neighbors in radar range as input. It filters all neighbors, that violate hard restrictions like being to far away, exceeding the maximum platoon size, or being blocked by a high vehicle density in between.
- **update happiness:** Among the output of **find candidates**, this method choses one candidate to update its happiness relation. This decision is done via different algorithms implemented inside this method. After choosing, the happiness relation is updated.
- **select best candidate:** After each updating process, the candidate with the currently highest happiness in relation to the actual vehicle is chosen. If the happiness of that candidate fulfills several constraints, this candidate is returned as a better match. Constraints are, that the candidate is not the own platoon leader and the happiness relation to the possible candidate in relation to the current relation to its own platoon leader is at least higher than the *platoon switching threshold*.
- **selection algorithms:** For each algorithm, there is an extra method implemented. The algorithms differ in the selection process. Depending on the algorithm, factors are the number of times a candidate was chosen, the current happiness relation and also randomness.

3.3.2 Vehicle Types

In order to simplify calculations about distances inside a platoon, we assume a standard length of a car. Modeling different types of cars, such as trucks, pick-ups, etc. is not necessary, since it has no direct effect on the platooning. Different desired speeds are important. We choose this for each car. Therefore the Federal Highway Research Institute of Germany [10] offers long term studies. Over a period of four years, they collected data at 60 measurement stations all over the highway network of Germany. The results are categorized into various subsections, e.g. speed

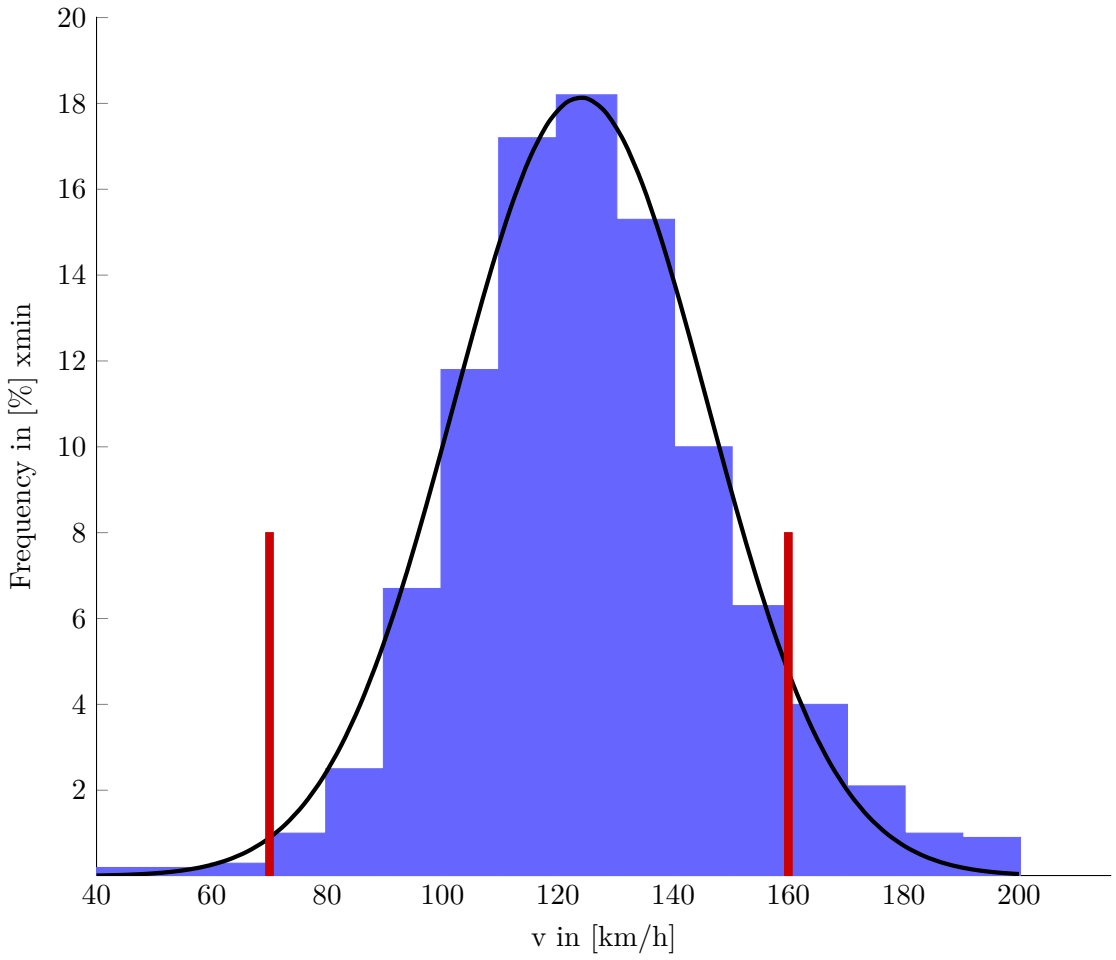

Figure 3.4: Speed distribution of vehicles on the German Autobahn from the Federal Highway Research Institute of Germany [10]. The data was collected from the year 2010 until 2014 at 60 measuring station located all over Germany. The histogram lists the relative measurement in bins of 10 km/h each. The black curve is a Gaussian approximation. The red bars symbolize the speed cutoffs of our simulation.

on highways with speed limit or speed on highways with three or more lanes. Figure 3.4 shows the average distribution of speed on German highways with no speed limit. We can approximate that distribution with a Gaussian curve parameterized by $\mu = 124$ and $\sigma^2 = 22$. Since approximately 90% of all data is in the range of $70\frac{km}{h}$ and $160\frac{km}{h}$, we cut off the speed distribution there. We can assume this as a speed limit, where most of the drivers with a higher desired speed stick to the rules and stop accelerating at a speed of $160\frac{km}{h}$. The amount of cars below the lower limit is negligible.

PLEXE calculates the desired speed as the product of the maximal allowed speed on the current road and the individual speed factor of a car. We generate the desired speed using a pseudo random number generator based on the distribution in figure 3.4. The speed factor is the ratio between generated speed and maximum speed.

3.3.3 Road Arrangement

One key research question is how to arrange a suitable test bed for the platoon simulation. In general there are two different systems - a highway with several exits on its way and a circle, where a platoon drives. Intuitively, a highway with start and end points seems to be the more realistic scenario. We compare different aspects in order to decide between both scenarios.

As a matter of fact, it is highly complicated to implement both systems. Knowing the distance between two vehicles is essential for a lot of mechanisms. Streets in SUMO are created on the basis of a coordinate system. In a straight highway scenario we can calculate the distance between two points directly. In a circle we can calculate the distance using polar coordinates. Since SUMO approximates circular streets with small straight ones [27], there is a small gap between calculated and real distance. This error grows proportionally with the radius. Another major difference is the cache implementation. Inside the cache of a vehicle, current information about candidates to form a platoon with are stored. Due to exploration, the happiness relations lose uncertainty. Unused information is overwritten after some time. In a circle, a faster platoon may approach from behind again, while it

is still inside the cache of a vehicle from a previous encounter. Happiness changes may remain undetected because that platoon was explored enough before. An extra detection is required, to recognize this happening and updating the cache.

Apart from technical issues, we take a closer closer look on the entire system. For measuring platooning behavior, we need a certain vehicle density. In a straight highway scenario, due to a restricted amount of vehicles, this system dispenses the cars widely depending on their speed. After an initial platoon phase, the ongoing platoon change rate decreases. When vehicles leave the highway, thus reducing the amount of cars, new vehicles are appearing. This leads to different clusters of cars with a lot of space in between. The highway situation must be created in a way, that a continuous stream of cars floods the system. After an initial phase, vehicles leaving and joining the system compensate each other. As mentioned before, due to limited computation the amount of cars is limited. The challenge is to form a system with enough cars for observing platooning behavior after the initial phase while keeping the system small enough to handle the amount of cars. In a circle, situations continuously occur, where platoons meet other platoons or single cars. After some time, the system develops into one of the following directions: Either it converges fully into a static scenario or a lot of cars circulate between different platoons. These side effects are hard to measure.

Overall, both scenarios have positive and negative effects. Since we can hardly estimate single effects on the overall system behavior, it is recommendable to use a more realistic setup. Further research may focus on this question.

3.4 Dynamic Change of Agents in Platoons

As for now, we have a set of methods for forming and rearranging a platoon. Vehicles are spawning at specific starting points with different desired speeds (see section 3.3.2). It is now time to involve the platooning process. Mentioned in section 3.2.1, we create a dynamic scenario, where multiple vehicles join or leave a platoon, if a certain happiness for a pair of vehicles is given. In section 3.4.1 we introduce our happiness construct as well as present an environment for implementing different algorithms. For test purposes, we start with a simple greedy bandit as a

ground truth. After successfully testing this algorithm, we go on to several machine learning algorithms mentioned in section 2.6. The adaption and implementation are described in section 3.4.2. The baseline is the distributed approach of Heinovski and Dressler [14].

3.4.1 Modified Multi-Armed-Bandit Approach

The multi-armed-bandit needs a storage, where the algorithm can get information about success and certainties from. Therefore we implement a storage for important information. We call this the happiness table. This happiness is a combination of parameters mentioned in section 3.2.3. The biggest difference from the standard multi-armed-bandit described in section 2.5 is our dynamic system. The normal setup has a set of choices, which are static and is not changing over time. In our scenarios, both situations occur. Joining cars lead to a change of the platoon's desired speed, following a change of happiness in between a platooning car and any other vehicle outside this platoon. In addition, differences in speed between vehicles lead to an ever ongoing process, that from the view of any car, platoons and other single cars appear in sensor range, along with other cars or platoons leaving the sensor range. Typically, the bandit's exploit-explore trade-off is offering a higher exploration rate first, during the course of getting informations the rate decreases to a minimum. This goes hand in hand with decreasing uncertainty [29]. The system needs a detector for rising uncertainty during the process. When a new option comes into sensor range, the exploration rate is reset to a higher value. As a result the new option is also considered. Moreover, we do not want to store unnecessary data about platoons that have already vanished from sensor range.

In order to meet these requirements, we establish the algorithm environment in Figure 3.5. The heart of this setting is the happiness table. We implement this as a LRU-cache [3] with a limited amount of data stored inside. The idea of our multi-armed-bandit system is, to choose a specific candidate for platooning from the neighbor (based on the implemented algorithm), calculate the actual happiness and store it in the happiness table. There are two modes available. Either we slow down the system due to storing the mean value or we store the current value directly. This is realized by the weighing factors ω and σ. $\omega = 1$ and $\sigma = 0$ leads to sto-

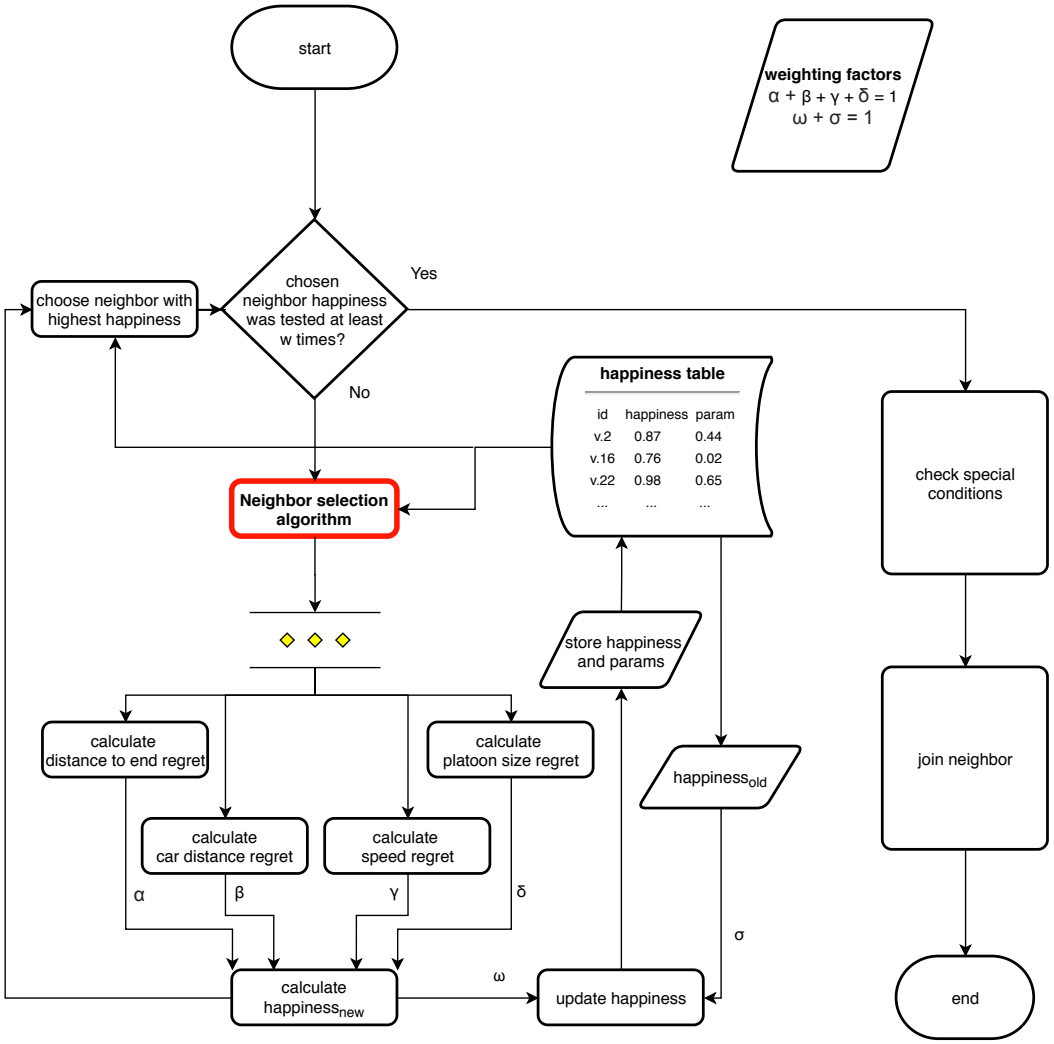

Figure 3.5: Flowchart of the algorithm environment. Greek letters represent a weighing factors. α, β, γ and δ are weighing factors for the happiness calculation. ω and σ are weighing factors for the moving average calculation. w is the *decision threshold*. A car has to be evaluated at least w times, before the car may be chosen as best candidate for platoon formation. The happiness table is a storage. Values are written into or read from that table.

ring the current value directly. According to the parameters, the stored happiness value changes rather faster or slower. We repeat this process over a certain time. After each calculation the supposedly best fitting neighbor is selected. The *decision threshold* w decides, how to proceed. If the best fitting neighbor was chosen less than w times, another calculation proceeds. Else the process checks special conditions, which may prevent a car from joining its best fitting neighbor. This can be a threshold, for example the relation between searching car and best fitting neighbor must have at least a happiness higher than the threshold, or the desired car to form a platoon will leave the highway immediately. The red marked box is the place to implement the selection algorithm. In the end, we want to have several matching algorithms, replace each other at that position in the flow chart. For ground truth, we select the distributed approach from Heinovski and Dressler [14]. Possible candidates are all cars in the sensor range. We define three restrictions. First, platoon members always present the happiness between inquiring vehicle and platoon leader. Second, a vehicle is only searching for candidates in front of it. We assume, that the happiness between an inquiring car and a neighbor is the same as the other way round. Therefore, we do not need to check that happiness twice. Third, when a new vehicle comes into sensor range, which is not in the happiness table, the happiness is always measured. Effects on the algorithm may occur and are detected through that routine.

3.4.2 Adaption of Machine Learning Algorithms for Platooning

Referring to figure 3.5, the neighbor selecting algorithm is the part of the test bed, where a specific algorithm is implemented. The algorithm has the current car and a list of candidates as input parameters. From that list of candidates, it chooses the best fitting candidate for updating the happiness relation from the car's point of view. In the following, we present the pseudo code implementation of machine learning algorithms introduced in section 2.6. On top, the distributed approach of Heinovski and Dressler [14], analysed in section 3.1, is implemented. Every car has a storage named *happiness table*. Happiness relations to other cars are saved inside it. Some algorithms need specific extra data to store. Therefore the *happiness table* saves a set of parameters, varying from algorithm to algorithm. In the end, we present the neighbor selection algorithm. This is a simple greedy algorithm, choosing

the candidate with the currently highest happiness. We have to differentiate - the machine learning algorithms choose a best fitting candidate in order to update the current happiness relation. The return of a best fitting candidate for platooning is separated and only works, if certain conditions are fulfilled.

ε - Greedy

```
1  constants: E_GREEDY_E
2  input: car, candidates
3
4  happiness table ← get_happiness_table_of(car)
5
6  if random() < E_GREEDY_E then:
7    chose best_neighbor randomly
8
9  else do:
10   def: best_value ← 0
11   def: best_neighbor
12   for candidate in candidates do:
13     if candidate not in happiness table then:
14       calculate happiness to candidate
15       store happiness in happiness table
16     end if
17
18     if get_mean_value_of(candidate) > best_value then:
19       best_neighbor ← candidate
20       best_value ← get_mean_value_of(candidate)
21     end if
22   end for
23 end if
24
25 calculate happiness to best_neighbor
26 store happiness and mean in happiness table
```

The ε-greedy algorithm is the most basic algorithm of the machine learning adaption. It is separated into two parts. From line 6 to line 21, the best candidate is selected. This is done randomly by a probability of the name-giving ε. Otherwise, the algorithm iterates over all candidates and selects the one with the highest mean value. A candidate, who does not have an entry inside the *happiness table* gets an initial calculation. This proceeds from line 13 to 16. When a candidate is chosen, in the second part we calculate the current happiness of that candidate and store

this and the mean inside the happiness table. To clarify, the mean value is used by ε-greedy to choose the to-be-updated candidate. The direct happiness value is stored for use in the best candidate selection algorithms.

UCB1

```
1  input: car, candidates
2
3  happiness table ← get_happiness_table_of(car)
4
5  def: best_value ← 0
6  def: best_neighbor
7  def: amount_tries ← 0
8
9  for candidate in candidates do:
10   amount_of_tries ← amount_of_tries + get_tries_of(candidate)
11
12   if candidate not in happiness table then:
13     calculate happiness to candidate
14     store happiness in happiness table
15   end if
16
17   happiness_value ← get_happiness_to(candidate)
18   heuristic_value ← sqrt( 2 log(amount_tries + 1) / (tries_candidate + 1) )
19
20
21   if happiness_value + heuristic_value > best_value then:
22     best_neighbor ← candidate
23     best_value ← happiness_value + heuristic_value
24   end if
25 end for
26
27 calculate happiness to best_neighbor
28 store happiness in happiness table
```

Upper confidence bounds (UCB) uses heuristics. In contrast to ε-greedy, the to-be-updated candidate is determined by the sum of happiness and heuristic value. This determination process from line 9 to 25 is the same as ε-greedy uses. Between different UCB algorithms, the heuristic value differs as mentioned in section 2.6. In UCB1 a quotient, containing the amount of how many times a certain car was chosen and the amount of how many steps this algorithm has taken until now, serves for

the desired behavior. The more a certain vehicle is explored, the more confident we are in our decision. The heuristic value drops. Otherwise, while a car's amount of explorations stucks and other vehicles are explored, its' heuristic value grows. The updating process is similar to ε-greedy. Since we do not need any mean value, this value is not stored inside the happiness table.

Bayes UCB

```
1  constants: VARIANCE_FACTOR
2  input: car, candidates
3
4  happiness table ← get_happiness_table_of(car)
5
6  def: best_value ← 0
7  def: best_neighbor
8
9  for candidate in candidates do:
10   if candidate not in happiness table then:
11     calculate happiness to candidate
12     store happiness in happiness table
13   end if
14
15   mean = ← get_happiness_to(candidate)
16   variance = ← get_variance_to(candidate)
17
18   if mean + VARIANCE_FACTOR *√variance >= best_value then
19     best_value ← mean + VARIANCE_FACTOR * √variance
20     best_neighbor ← candidate
21   end if
22 end for
23
24 calculate happiness to best_neighbor
25 calculate mean and variance to best_neighbor
26 store happiness and mean and variance in happiness table
```

This is another heuristic based algorithm. Bayes USB assumes a Gaussian distribution behind the samples. We do not expect the platooning problem to be Gaussian, but we want to see how this suboptimal algorithm will behave under the given conditions. The mean and variance is stored for every car. By picking the candidate with the highest possible value (mean $+VARIANCE_{FACTOR} * \sqrt{variance}$), the algorithm proceeds to do a moving mean and variance calculation over all samples of

a vehicle relation. The current happiness value is just stored for the best candidate selection routine, the same reason as in UCB1. Updated mean, updated variance and current happiness are stored.

Thompson Sampling

```
1  input: car, candidates
2
3  happiness table ← get_happiness_table_of(car)
4
5  def: best_value ← 0
6  def: best_neighbor
7
8  for candidate in candidates do:
9    if candidate not in happiness table then:
10     calculate happiness to candidate
11     initialize α–β–distribution
12     store happiness and distribution in happiness table
13   end if
14
15   get sample of distribution
16   if sample >= best_value then:
17      best_value = sample
18      best_neighbor = candidate
19   end if
20 end for
21
22 calculate happiness to best_neighbor
23 update distribution
24 store happiness and distribution in happiness table
```

This algorithm is different. From line nine to 12 a new α-β-distribution is set up for candidates with no prior knowledge. We choose an initial setup of $\alpha = 0.5$ and $\beta = 0.5$. For selecting the to-be-updated candidate, from every distribution of each car, we sample once and choose the highest sample. The vehicle with the highest sample is updated afterwards. We calculate the happiness for this vehicle. With that information we can update the distribution to be more reliable. Current happiness, α and β are stored. Contrary to all other algorithm, this one does not prefer higher values (as greedy does) or values with higher uncertainty (as UCB does). Instead, the sampling leads to a small chance, that also worse options are explored again.

This may correct errors from initially bad samples. On the other side this produces overhead, so this algorithm may take more time for decision making. This behavior makes it also unclear, how this algorithm performs. As mentioned by Russo et al. [26], this algorithm works fine on many problem setups.

Heinovski (Distributed)

```
1  input: car, candidates
2
3  happiness table ← get_happiness_table_of(car)
4
5  def: best_value ← 0
6  def: best_neighbor
7
8  for candidate in candidates do:
9    calculate speed deviation to candidate
10   calculate distance to candidate
11   happiness ← 0.6 * speed deviation + 0.4 * distance
12   if happiness >= best_value then:
13     best_value = happiness
14     best_neighbor = candidate
15   end if
16 end for
17
18 return best_neighbor
```

This is a reimplementation from Heinovski and Dressler [14]. This results in a major change in the happiness calculation. No storage is used. From a certain starting point, all vehicles calculate their happiness to all neighbors in sensor range. The happiness is calculated in line 11 and differs a lot from our introduced happiness. There is only one iteration. When a best candidate is chosen, the vehicle immediately tries to match with it as a platoon. There is no decision time. Otherwise, there is also no platoon regrouping, since platoon changes are not allowed. There is one major adaption of that algorithm. In the original scenario, there is no car leaving or entering the highway after the platoon formation process has started. This is modified in order to allow cars to leave the highway. Entering cars are looking as soon as possible for a matching platoon. Due to that, we expect the algorithm to deliver bigger platoons after the first exits, than mentioned in the publication of Heinovski and Dressler [14].

3.4.3 Candidate Selection

While the previous section has dealt with the happiness update process, the candidate selection process is the same for all algorithms.

```
1  constants: PLATOON_CHANGING_THRESHOLD, DECISION_THRESHOLD, RADAR_DISTANCE
2  input: car, candidates
3
4  def: best_value ← 0
5  def: best_neighbor
6  happiness table ← get_happiness_table_of(car)
7
8  for candidate in candidates do:
9    happiness_value ← get_happiness_to(candidate)
10
11   if happiness_value > best_value then:
12     best_value ← vehicleInfos[car].get_happiness(candidate)
13     best_neighbor ← candidate
14   end if
15 end for
16
17 if best_neighbor is None then:
18   return None
19 else if best_neighbor is own platoon leader then:
20   return None
21 end if
22
23 if MODE is not HEINOVSKI then:
24   calculate happiness to best_neighbor
25   if happiness_value + PLATOON_CHANGING_THRESHOLD > best_value then:
26     return None
27   else if get_tries_of(candidate) >= maximum of (DECISION_THRESHOLD,
       2.5 * amount_candidates) then:
28     return best_neighbor
29   else:
30     return None
31   end if
32 else:
33   return best_neighbor
34 end if
```

In the for loop from line nine to 15, among the candidates, the candidate with the best stored happiness is chosen. It may occur, that there is no candidate at all in radar distance. Even so, the best candidate may actually be the current leader. In both cases, no changing is desired, so we return None. From line 23 to 36, special

conditions dependent on the currently performing algorithms are checked. For all algorithms except Heinovski [14], the happiness value of the best candidate has to be at least a certain amount higher, that the value of the current leader. This amount is regulated by the variable *platoon switching threshold*. If this condition is passed, we check, if the candidate was explored a certain amount of times. This must be at least the maximum of *decision threshold* times and 2.5 times the amount of candidates in sensor range. With this condition, we want to guarantee longer decision processes, if there are more candidates to choose from. If the selected algorithm is the approach from [14] (line 32), then there is no further check required.

3.5 Expected Problems

We now have a highly dynamic system with a lot of parameters to modify. Depending on these parameters, we get a wide spectrum of results. Nonetheless there are some challenges we have to deal with. We keep them in mind and try to minimize their influence.

First of all, there is the question of how the given highway influences the results. We have a highway length 22 kilometers. With three entries / exits in between, we have phases of about 6 kilometers, where platoons form without the dynamic influence of cars leaving and joining. If we try another highway setup, the results are massively be influenced by that. A comparison is very difficult. Therefore, we use the same highway for all simulations. Since there are a lot of other parameters to modify, we can keep this source of noise out.

Second, how many vehicles do we need for a stable system? If the vehicle density on a certain part of the road is high, we have more platoon formations than on a part with a low vehicle density. For results with a most homogeneous density, we need to spawn vehicles at the beginning in repeating time periods. Furthermore, on entries we need to spawn vehicles as well. An indicator is the amount of leaving cars at that exit. We can compensate this amount by spawning cars there. In order to guarantee this behavior, we need a huge amount of vehicles. Because it is only a simulation of distributed systems, the computation behaves like a centralized system. Due to neighbor relation calculations, the computation time goes in $O(n^2)$.

In order to meet a good compromise between platooning behavior and computation time, we select 250 cars as the amount to go.

Third, since we do not have any cars on the highway at the beginning of the simulation, we have a specific settling time of the system. During this time, the system may behave differently than afterwards. On the other hand, it takes times to spawn 250 independent cars, as well as replacing the leaving cars with new ones. Before the 250th car has spawned, the first vehicles leave the highway. If we let vehicles appear at the highway entries from the beginning, we get separate groups of cars, which are not interacting with each other. We have to achieve a balanced state system with limited computational power and limit simulation time for each parameter setting.

Finally, we go deeper into dynamic platoon switching. If a vehicle detects another car with a better happiness relation than the current one to its own leader, it changes platoons. Because a vehicle switches without acknowledgment of the new leader's side, the platoon change may behave in an egoistic way. While the single, platoon switching car improves its happiness, the happiness of all platoon members decrease a bit. If this takes several times, we may get a system, where vehicles circulate from one platoon to another. A joining car decreases the overall happiness, this results in another platoon change, etc. A circulation system has great costs for platoon forming and is not desirable, furthermore such a live lock inhibits the overall happiness extremely.

4 Results

This chapter shows the results of different simulations. We have a wide variety of parameters that determine the resulting behavior. We define a scenario as a set of parameters. For every parameter setting, 12 simulations with random seeds $\in [0, 11]$ are tested. Choosing the same seed guarantees the same output of the pseudo random number generator. Thus leading to comparable scenarios among the different algorithms.

We simulate each scenario with five different algorithms: ε-greedy, UCB 1, Bayes UCB, Thompson-Sampling (introduced in section 2.6) and the approach of Heinovski and Dressler [14]. Table 4.1 represents all used scenarios with parameter setup.

First of all, we start with an overview about the behavior of different simulations in section 4.1. Besides changing parameters, there are two different highway scenarios which are also tested - on the one hand, there is a *start to end scenario*, where all vehicles will appear at the start of the highway and will drive until the end of the highway of 22 kilometer total length. There are no entries or exits in between. On the other hand, we test a *dynamic scenario*. There are three entries and exits in between, where vehicles will randomly join and leave the highway. The highway is shown in figure 4.1.

Section 4.2 deals with a comparison between both setups. Section 4.3 takes a look at the expected problems, described in section 3.5 of a previous chapter. Finally, we compare multiple tested scenarios. We get an idea of how well our implemented algorithms are working in different parameter settings. As mentioned above, all scenarios differ depending on the chosen parameters. We give a short overview about the used parameters influencing the behavior. All parameters affect the algorithms ε-greedy, UCB 1, Bayes UCB and Thompson-Sampling. Heinovski is not affected

Scenario Label	**Dynamic (D)** or **Start to End (SE)**	**w**	**t**	**speed up active**	**weighting params** $\alpha, \beta, \gamma, \delta$
A	D	5	0.01	Yes	1,1,1,2
B	D	5	0.01	Yes	1,1,2,1
C	D	5	0.01	Yes	1,1,1,1
D	D	5	0	Yes	1,1,2,2
E	D	5	0.04	Yes	1,1,2,2
F	SE	5	0	Yes	1,1,2,2
G	SE	5	0.04	Yes	1,1,2,2
H	SE	5	0	No	1,1,2,2
I	SE	5	0.04	No	1,1,2,2
J	D	5	0	No	1,1,2,2
K	D	5	0.04	No	1,1,2,2
L	SE	10	0	No	1,1,2,2
M	*SE*	*10*	*0.04*	*No*	*1,1,2,2*
N	*SE*	*5*	*0.01*	*Yes*	*1,1,2,2*
O	*SE*	*5*	*0.02*	*Yes*	*1,1,2,2*
P	*D*	*5*	*0.01*	*Yes*	*1,1,2,2*
Q	*D*	*5*	*0.02*	*Yes*	*1,1,2,2*

Table 4.1: Overview of the parameter setting of each scenario. This table is referenced by most figures of these chapter. The parameters are described below. Scenarios above the double line are used for the box plot in section 4.4. The scenarios below are mentioned in this section, too. But they are not used in section 4.4. *Cursive scenarios* are found in the appendix.

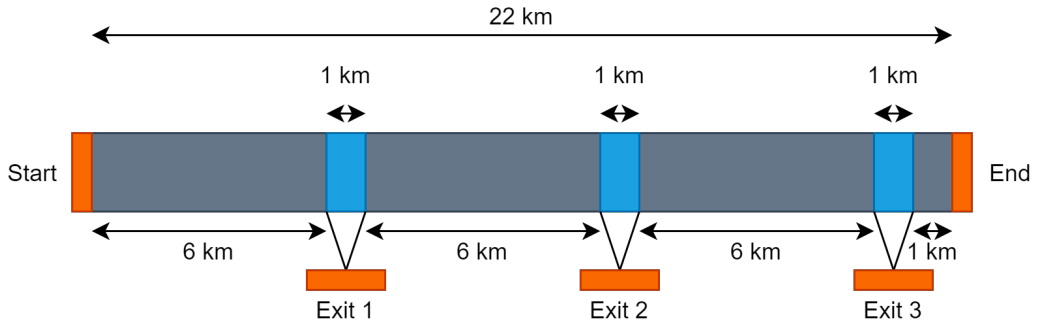

Figure 4.1: Outline of the highway. Orange marks positions, where vehicles appear or disappear. Blue marks highway entries and exits.

by any of these parameters.

- The **decision threshold** w is the amount of times a vehicle must evaluate a candidate before the vehicle is allowed to join the candidate. At a certain time step, if a neighbor is the best option for matching, the searching vehicle must have evaluated this neighbor at least w times. In that case the processing algorithm returns the candidate as best matching candidate.

- The **platoon switching threshold** t is a value for reinforcing and weakening the occurrence of platoon changing of a vehicle. If a car detects another vehicle as best matching candidate, the car only leave its current platoon, if the improvement of happiness of the new platoon leader in comparison to the current one is at least higher than t. A higher value lead to less changes, while a low value will reinforce platoon changing.

- The **speed up** value modifies the start conditions. Our system prefers platoons over single cars. This means, that driving in a platoon is always better than driving alone. If the speed up is activated, all vehicles will form a platoon as soon as they recognize another car in range. Afterwards, the normal algorithms deciding procedure starts. If the speed up is deactivated, the normal deciding procedure is activated from the start.

- The **happiness** is separated into four happinesses components. All of them have a weighting factor. There are the distance to end weighting factor α, the distance between a desired platoon leader and the current position weighting factor β, the speed difference weighting factor γ and the platoon size weighting factor δ. The weighting factors are chosen in $\mathbb{R}^+$. All values are automatically normalized, so that $\alpha + \beta + \gamma + \delta = 1$.

This chapter focuses on box-plots. All plots are build on the same schema. Each evaluated algorithm always has the same color. The parameters are written in the figures' captions. If the amount of seeds deviates from the standard, the parameter n presents the amount of seeds used for the specific scenario. All algorithms have a maximum simulation time of ten minutes in real time. We have troubles with simulation crashes shown in section 4.1. Due to errors, some simulations may end

earlier. We use *NumPy 1.18* as the statistical framework. All box plots are generated by the standard parameter setup. All error bars are showing the standard deviation.

4.1 Implementation and Simulation

For testing platoon formation algorithms, a well running simulation is necessary. This means, that all maneuvers must work properly, like joining, leaving and merging. The active algorithm just should decide, whether to form a platoon with a certain car or not. The technical mechanisms behind forming a platoon are not relevant for those algorithms. Unfortunately, it was not possible to fix all edge cases in maneuvers. This results in some crashes of vehicles. A routine detects crashes and removes the involved cars from the highway. This allows the simulation to go on without being disrupted by an accident, because following vehicles would stop. Removing crashed cars sometimes leads to errors in SUMO. They are reproducible, but the reason for these errors was not detected. To achieve comparability, we require simulations to go on as long as possible.

We test the occurrence of crashes the average run time of different scenarios. As a baseline, a scenario without platoon formation with 250 cars is given. All cars get a randomly selected destination and will appear at the beginning or at an entry. They will leave the highway at one of the exits or will drive on until the end of the highway. All of these simulations end after the maximum simulation time of ten minutes. No crashes or simulation aborts are detected. There are no platoon changes and merges at all. Further on, in various scenarios we observe the average simulation time, amount of crashes and amount of platoon changes and merges. While vehicles are changing or merging a platoon, there is a chance, that the maneuver will brake up due to external factors. We also observe the number of these interruptions.

In figure 4.2 scenario D from table 4.15 is shown. It is *dynamic scenario* with *decision threshold* $w = 5$ and *activated speed up*. The amount of changes is high due to the switching threshold $t = 0$. This means, that a car will always change its platoon, if the happiness inside the new platoon is better - even if the possible improvement is small and the risk of aborting does not justify this switch. Cars will drive a randomly selected route. Starting points are the start of the highway and all

three entries. Ending points are all three exits and the end of the highway. With a chance from approximately 22 to 32 percent, a changing maneuver fails. The merging quote is less than ten percent of the changing quote. A merge only occurs, when a platoon leader decides to change platoons. Nearly all merging maneuvers succeed. Comparing the algorithms, we identify UCB 1 as the algorithm with the highest dynamics by far. Thompson-Sampling is second, but the distance to $\varepsilon-$greedy is small. Bayesian UCB follows with low dynamics, while the Heinovski approach does not have any dynamics. Referring to the simulation time, we can identify, that all algorithms have a mean time of 462 to 540 seconds. The standard deviation is between 80 and 140.

In contrast, figure 4.3 shows scenario G from table 4.15. It is a conservative *start to end scenario* with $w = 5$ and *activated speed up*. The switching threshold $t = 0.04$ leads to even less changes, because a car will only change if the new platoon will improve its own happiness by at least 4 percent. All cars will start at the beginning of the highway and will drive until the end of the highway. No entry or exit is used. This conservative scenario leads to less crashes, changes and merges. The crashes reduce by more than half. The decrease of the amount of changes is dependent on the algorithm. Even a dynamic approach like UCB 1 reduces its changes by about more than half. Thompson-Sampling and $\varepsilon-$greedy decrease the amount of changes by about a third. The amount of aborted changes decreases as well, but stays at a high level in comparison to the other scenario. Merges are reduced by approximately a third, while nearly all merging maneuvers succeed. Referring to the simulation time, the mean time is between 508 and 600 seconds. Except for UCB 1, there is a small standard deviation, less than 40 seconds.

We study this behavior with the parameter $t \in [0, 0.01, 0.02, 0.04]$ in the *start to end scenario* and in the *dynamic scenario*. Generally spoken, a lower t leads to more platoon changes. More platoon changes lead to more crashes and more crashes lead to a lower simulation time. The comparison between the *start to end scenario* and the *dynamic scenario* lead to a similar conclusion. The *start to end scenario* leads to more platoon switches, thus leads to more crashes and a lower simulation time. This is the expected behavior, since a *start to end scenario* has no leaving and joining maneuvers are caused by highway entries and exits.

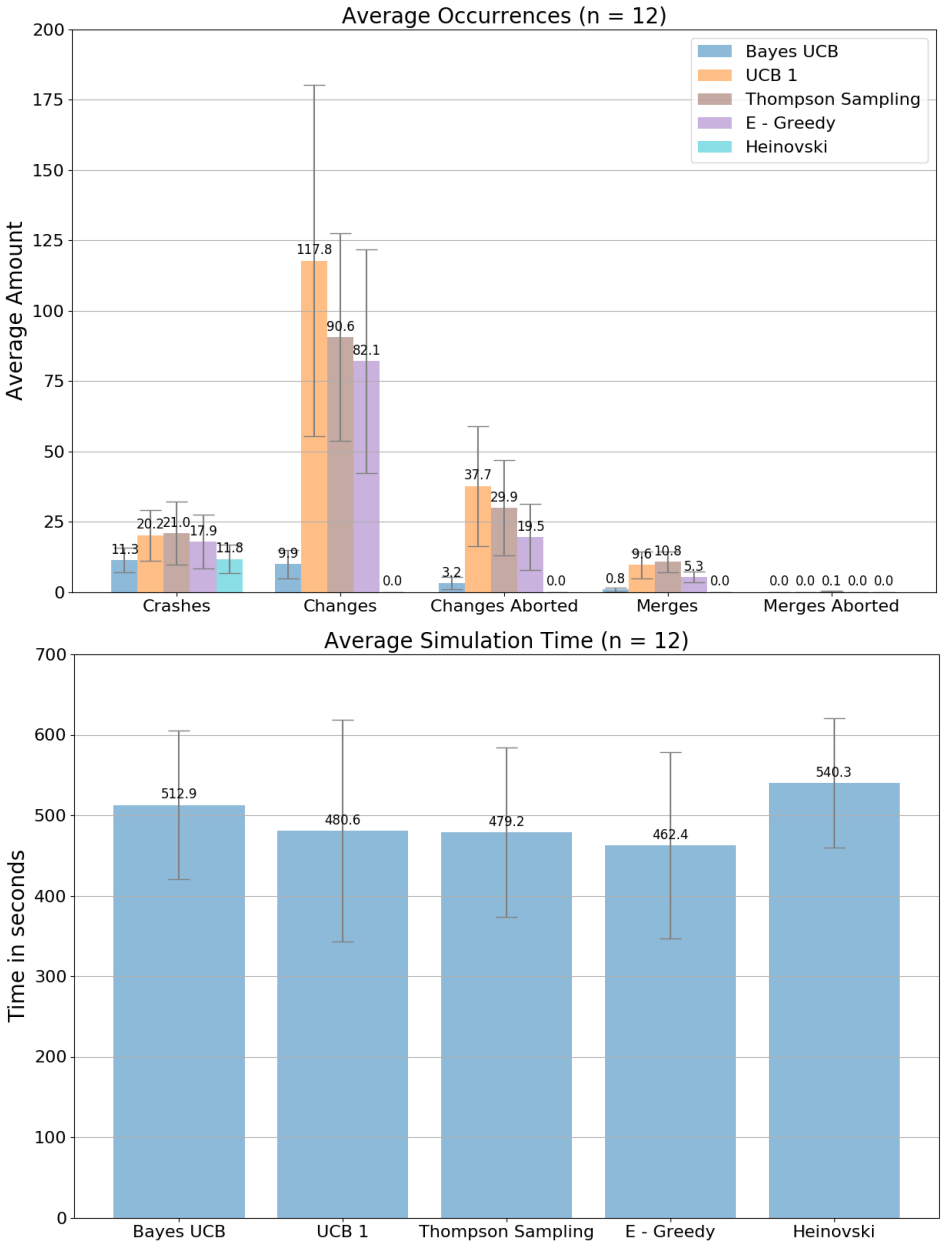

Figure 4.2: Scenario D from table 4.1. Average occurrences of events and average simulation steps before simulation crashes. Statistics generated by NumPy 1.18 with standard parameters: mean value, error bars are standard deviation.

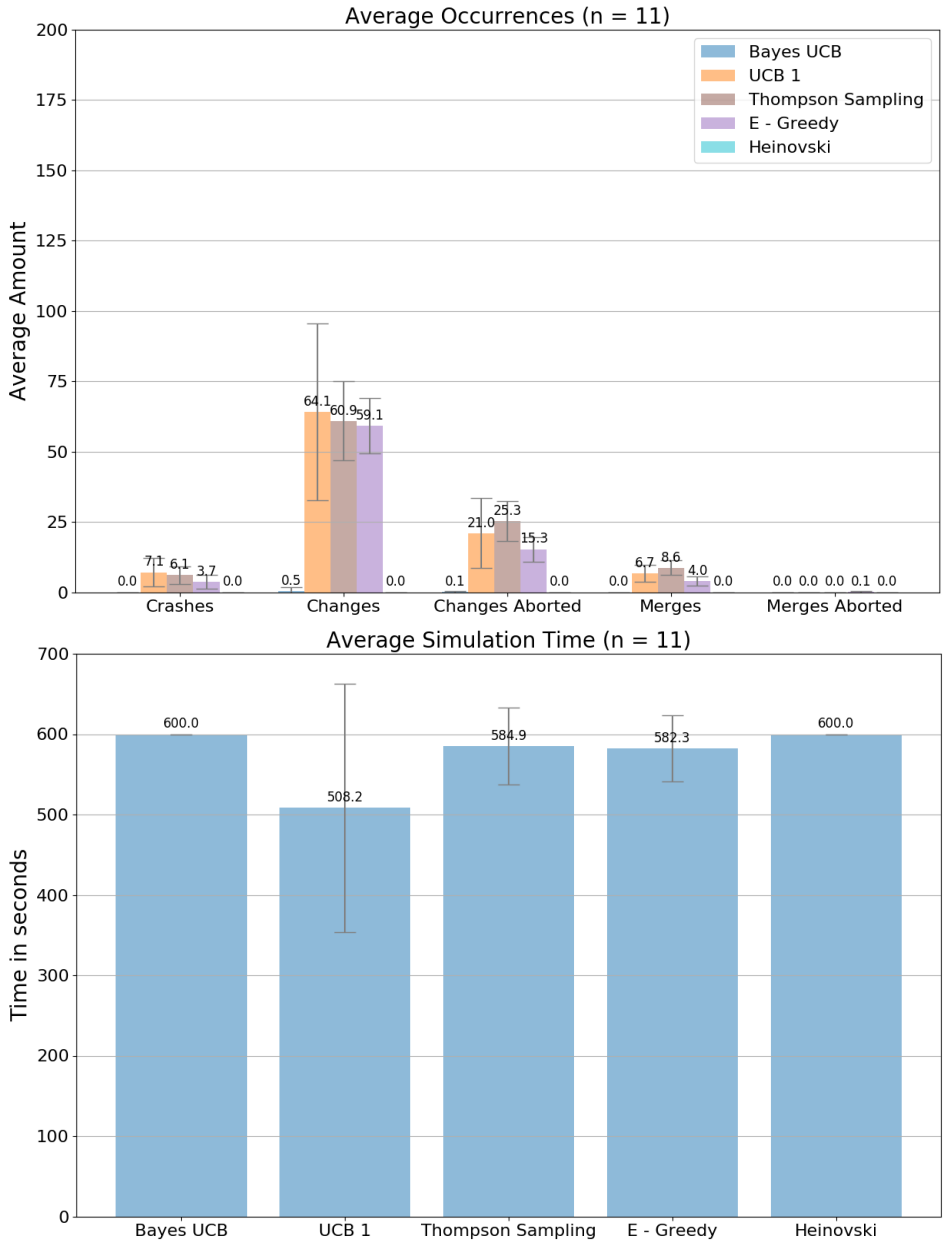

Figure 4.3: Scenario G from table 4.1. Average occurrences of events and average simulation steps before simulation crashes. Statistics generated by NumPy 1.18 with standard parameters: mean value, error bars are standard deviation.

4.2 Model Setup

Now we take a deeper look at the influence on the platoon size. To gain a better understanding of how the parameters influence the scenarios, we separate this section into two rudimentary different scenarios. Section 4.2.1 deals with the *start to end scenario* mentioned above. The *dynamic scenario* is researched in section 4.2.2. Our assumption is, that Heinovski will find platoons fast, which give an initial push to the average platoon size. Because Heinovski has no dynamic component, after an initial platooning phase, the algorithm sticks to the same average platoon size. The other algorithms are dynamic. They need more decision time, which will initially slow them down. A possibility to compensate this the introduced *speed up*. Due to platoon changing maneuvers, these algorithms get better over time. Getting better means, that the trade-off between happiness and platoon size is better balanced than in other scenarios. We have several weighting factors for happiness components. Section 4.2.3 observes the development of the happiness in regard to the platoon size. We test different weighting factor setups.

4.2.1 Start to End Scenario

Figure 4.4 shows two similar simulations in a *start to end scenario*. The platoon switching threshold is $t = 0$. No speed up is active. This means, that from the beginning, each car has to evaluate a candidate for platooning at least w times, before the algorithm may return this candidate as best neighbor. The upper plot shows the behavior with $w = 10$, while in the plot below $w = 5$ is set. This is the only difference. In the starting phase, the Heinovski approach outperforms all other approaches. Because there is no minimal decision time needed, the algorithm chooses its matches quickly. With $w = 10$, the curve of average platoon size grows slowly, while $w = 5$ grows a lot faster. Even with $w = 5$ the approach of Heinovski is slightly better. At simulation step 550, another behavior can be observed. While in the $w = 10$ simulation all algorithms except $\varepsilon-$greedy are worse than Heinovski, with $w = 5$ $\varepsilon-$greedy, UCB 1 and Thompson-Sampling catch up with Heinovski. Only Bayesian UCB is worse. This figure is exemplary for the behavior related to the decision threshold w. All simulations with parameter $t \in [0, 0.01, 0.02, 0.04]$ with or without speed up show the same dependence of w. The plots of the scenarios with $t = 0.04$ are in the appendix (scenario I and M from table 4.1).

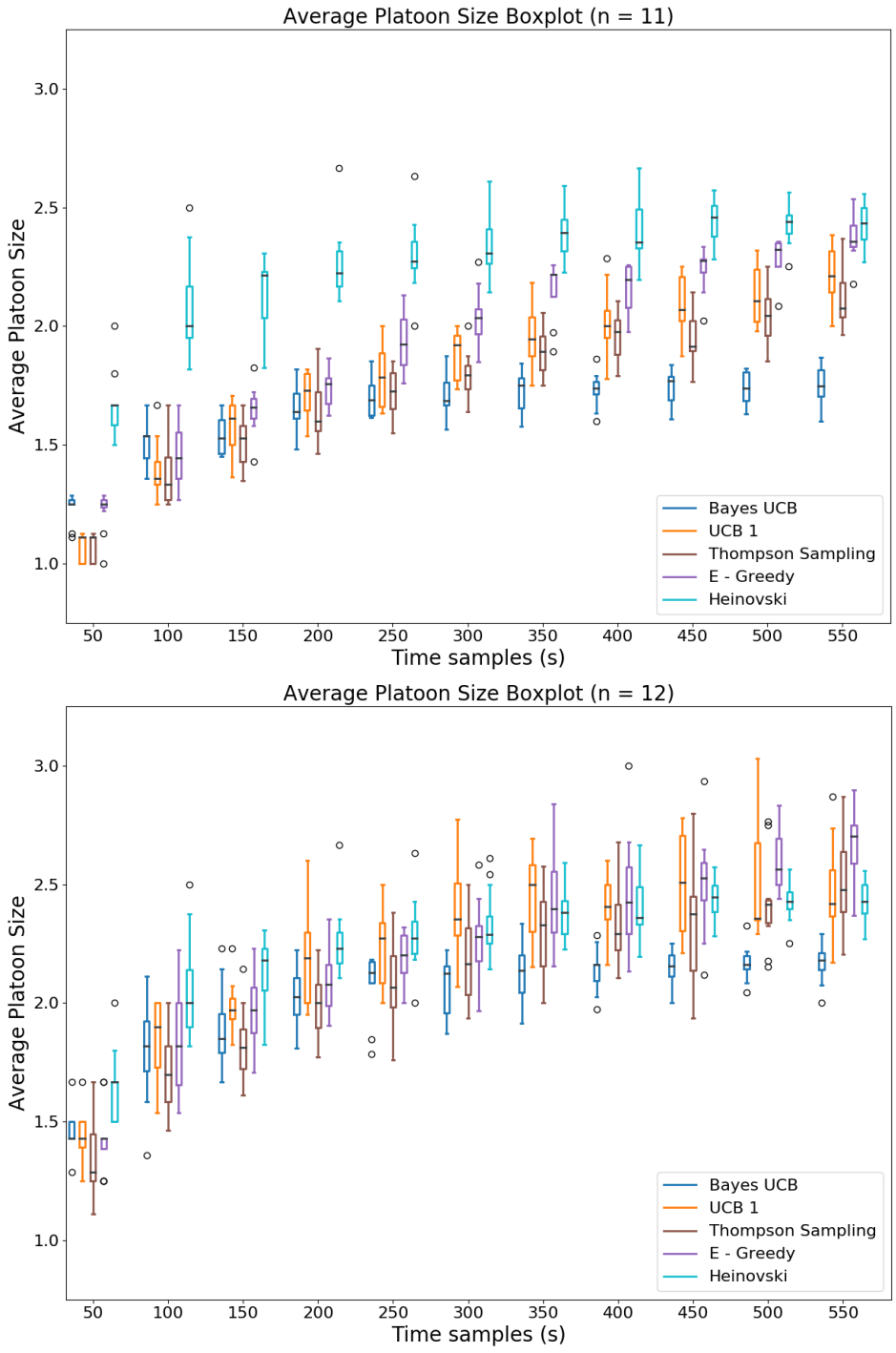

Figure 4.4: Scenario L (upper) and H (below) from table 4.1. Comparison of different decision thresholds w in a *start to end scenario*. Platoon switching threshold $t = 0$ is fixed. Speed up is deactivated. First diagram shows scenario with $w = 10$. The scenario below is simulated with $w = 5$.

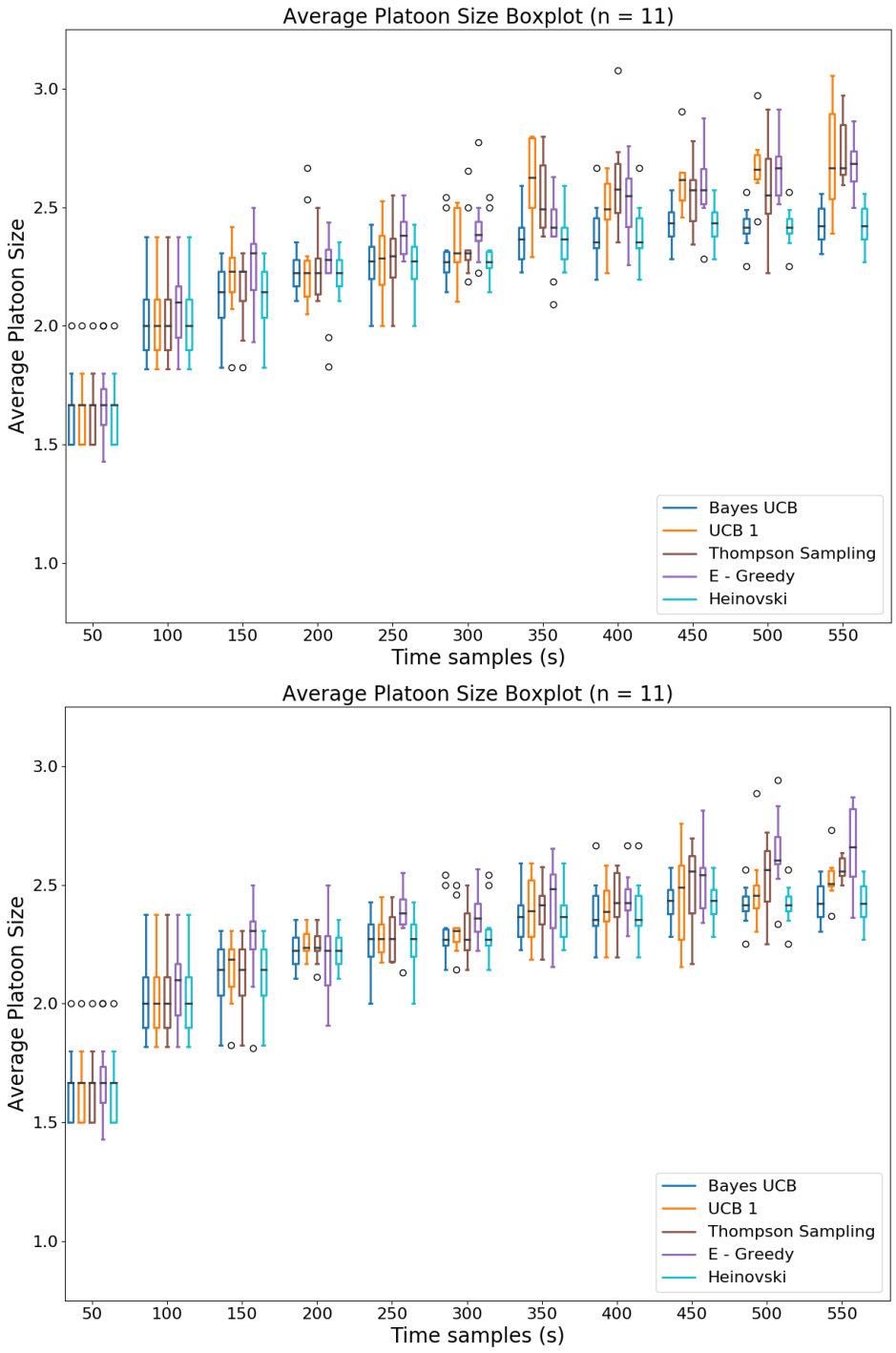

Figure 4.5: Scenario F (upper) and G (below) from table 4.1. Comparison of different platoon switching thresholds t in a *start to end scenario*. First diagram shows a scenario with platoon switching threshold $t = 0$ and the second one is simulated with $t = 0.04$.

In order to try to avoid the initial advantage of Heinovski, we activate the speed up. This option is assuming, that driving in a platoon is always better than staying a single car in our approach. When driving alone, a vehicle matches with the first available platoon candidate. Afterwards the algorithms will decide about platoon changing. We set $w = 5$ as fixed. Now we take a look at the platoon switching threshold t. In figure 4.5 the first simulation has parameter $t = 0$. The second one is simulated with $t = 0.04$.

In comparison to figure 4.4 we have the direct impact of the speed up after 50 and after 100 seconds. All algorithms have approximately the same average platoon size. A car is now immediately selecting the first neighbor for platooning, before the algorithm starts selecting candidates. This behavior is the same for all algorithms. Without speed up, the approach of Heinovski and Dressler [14] outperforms the other algorithms at first. Since it is the same behavior as before, the average platoon size of the algorithms ε-greedy, UCB 1 and Thompson-Sampling is growing. The speed up helps to form platoons faster, but the happiness decreases. This results in a higher average platoon size at the end of the simulation after second 550. The results of Bayes UCB are slightly better than before, but are falling behind in comparison to the other three algorithms. The results of Heinovski's algorithm are exactly the same because the speed up has no effect on the algorithm's behavior.

The effect of $t = 0.04$ leads to a system with less platoon changes. While the start is similar (until 100 seconds), the algorithms UCB 1 and Thompson-Sampling outperform the other simulations in the scenario with $t = 0$. ε-greedy and Bayes UCB behave similarly. Because Heinovski's approach does not have any platoon changing vehicles, this algorithm behaves exactly the same. We simulate with parameters $t \in [0, 0.01, 0.02, 0.04]$. $t = 0$ performs best, followed by growing values of t. $t = 0.04$ performs worst. The difference between consecutive values is not significant. The scenarios with parameter $t = 0.01$ and $t = 0.02$ are found in the appendix (scenario N and O of table 4.15).

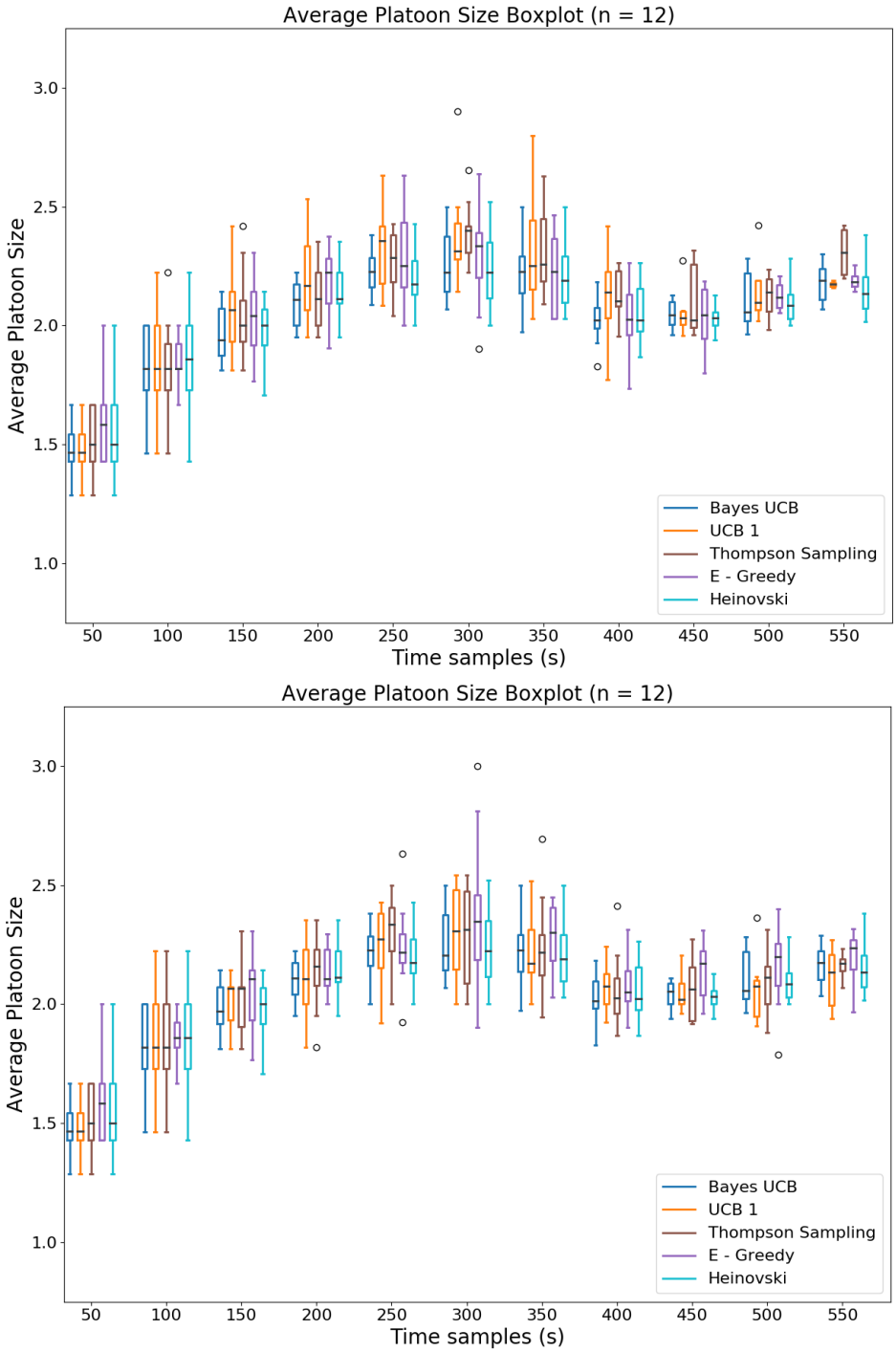

Figure 4.6: Scenario D (upper) and E (below) from table 4.1. Comparison of different platoon switching thresholds t in a *dynamic scenario*. First diagram shows scenario with platoon switching threshold $t = 0$. The scenario below is simulated with $t = 0.04$.

4.2.2 Dynamic Scenario

In this chapter, we focus on the influence of the platoon switching threshold t on system performance. Figure 4.6 shows two similar simulations in a *dynamic scenario*. The *decision threshold* is $w = 5$. Speed up is active. We simulate again with parameters $t \in [0, 0.01, 0.02, 0.04]$. The upper plot shows the behavior with $t = 0$, while in the plot below t is set to 0.04. In both simulations, the average platoon size is growing from the start. The first peak is at second 300. Afterwards the average platoon size is decreasing until second 450. Then, the average platoon size is growing again. Among all tested thresholds, there is no significant difference. For the Heinovski algorithm, the threshold t is not relevant. So it leads to exactly the same result in all simulations. The Bayes UCB algorithm also looks alike. Only at a detailed level, there are some minor differences. ε-greedy, UCB 1 and Thompson-Sampling show the biggest differences; but also these are not significant.

Besides, there are a lot of simulations not running until the end. Table 4.2 shows the amount of proceeding simulations at each time step for both platoon switching thresholds $t = 0$ and $t = 0.04$.

	Time Steps					
Algorithms	50s	150s	250s	350s	450s	550s
Bayes UCB	12	12	12	11	10	8/9
UCB 1	12	11	10/11	9/10	6/9	2/6
Thompson-Sampling	12	11	9/11	9	7	4
ε-greedy	12	12	12/11	10/11	7/10	4/7
Heinovski	12	12	11	10	8	6

Table 4.2: Amount of runs n at each time step used for the box plot diagram. If there are two numbers in a field, the first is the amount of simulations for platoon switching threshold $t = 0$. The second number is for simulations with platoon switching threshold $t = 0.04$. In fields with a single number, n is identical for both values of t.

With $t = 0.04$, we have at least seven simulations of all algorithms proceeding until second 450. In the end all algorithms except Thompson-Sampling have at least six proceeding simulations. With Thompson Sampling only four simulations

complete the entire process. With $t = 0$ we have the same amount of simulations for Thompson-Sampling, Bayes UCB and Heinovski at step 450 and at the end. ε-greedy decreases from $n = 7$ simulations at time step 450 to $n = 4$ at the end, while UCB 1 decreases from $n = 6$ to $n = 2$.

4.2.3 Average Happiness to Average Platoon Size Relation

We introduced a happiness as a complementary measurement to the platoon size in section 3.2.3. A high average platoon size leads to a higher number of cars, that use slipstream effects, thus saving fuel. The happiness relation between two vehicles is an indicator of how similar both cars are in terms of their happiness components. Besides the platoon size there are three more factors influencing the happiness: the maximum distance the car can stay in the platoon, which depends on the destination of the car and the other cars in the platoon, the distance between the joining and the to-be-joined vehicle and the speed deviation. In this section, we will test the effect of different weighting factors on the relation between happiness and platoon size. We are focusing on the speed deviation weighting factor γ and the platoon size weighting factor δ because we are expecting them to have the greatest influence on this system. All other parameters are the same in all simulations. The first plot shows the average happiness. The second plot shows the average platoon size. The current platoon size is a component of the happiness. By choosing extreme weighting parameters (for example $\alpha = \beta = \gamma = 0$) and only weighting the platoon size weighting factor δ, we get an self-enforcing system. Our objective is to form well-fitting platoons, thus formed by homogeneous cars. Such an extreme scenario does only lead to big platoons, without considering the homogeneity.

Figure 4.7 shows a simulation with weighting parameter setup $\alpha = \beta = \gamma = \delta = 1$. All parameters are weighted equally. This serves as a baseline.

At 100 seconds the happiness raises to approximately 0.75. The ε-greedy algorithm is slightly, but not significantly lower. Afterwards the average happiness of all algorithms stays in inside the interval $[0.7, 0.8]$. The happiness of ε-greedy catches up to the other algorithms. At 350 seconds a decrease begins. This ends at 450 seconds. Then, the happiness improves on a low level. Until second 100, there is a

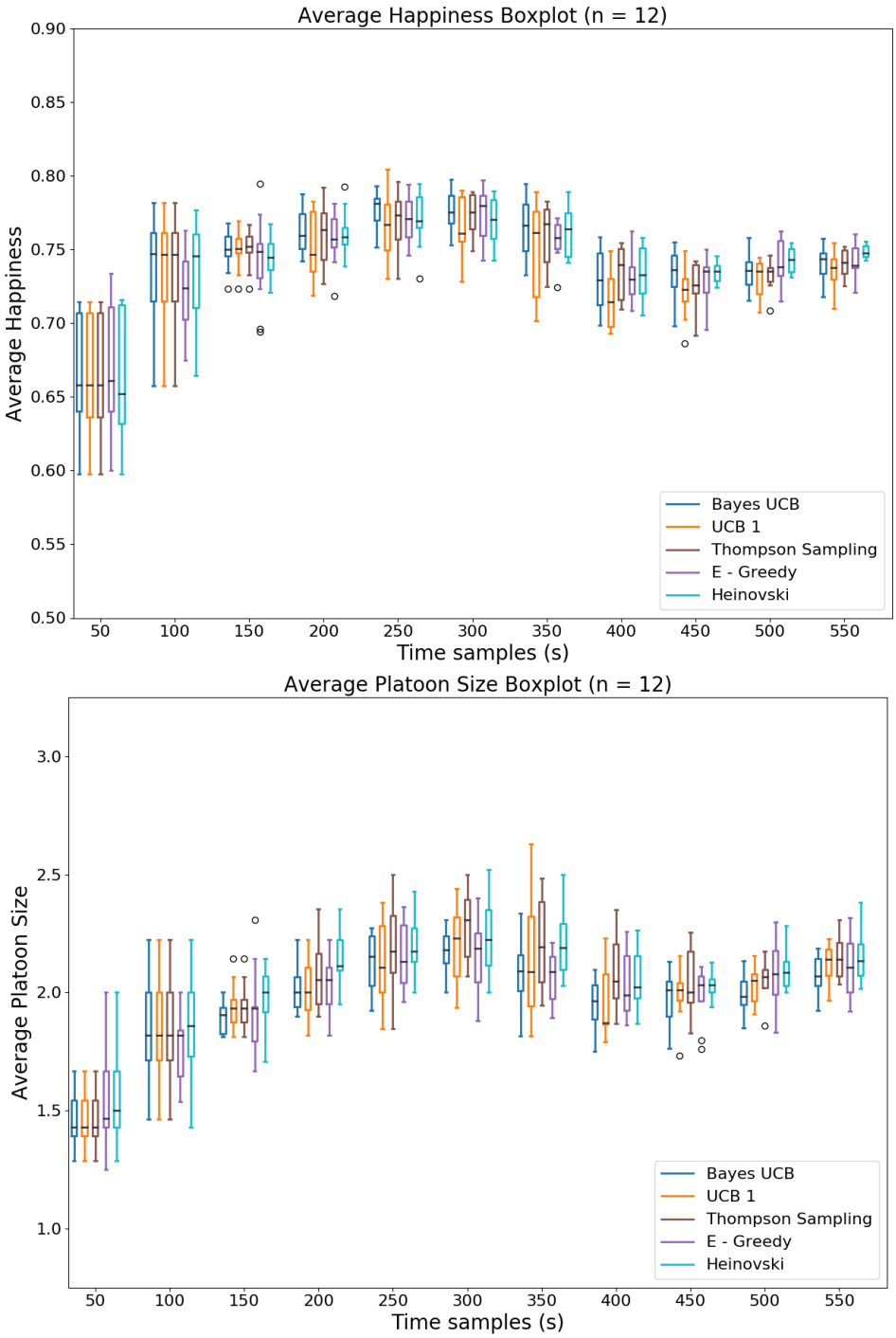

Figure 4.7: Scenario C from table 4.1. Average Happiness (upper plot) in comparison to the average platoon size (lower plot). *Dynamic scenario.* $t = 0.01$, $w = 5$, speed up active. Happiness parameters are: $\alpha = \beta = \gamma = \delta = 1$.

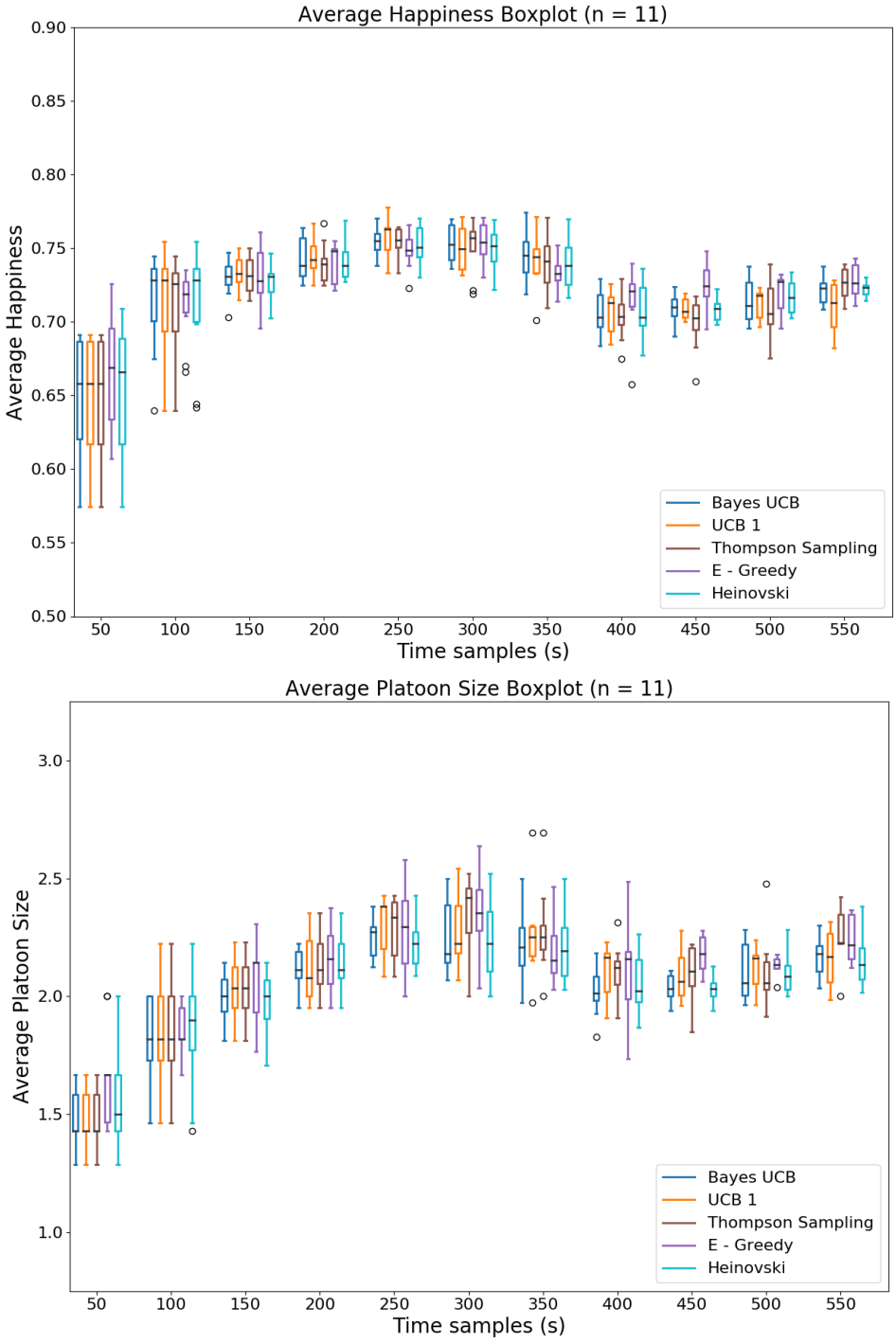

Figure 4.8: Scenario P from table 4.1. Average happiness (upper plot) in comparison to the average platoon size (lower plot). *Dynamic scenario.* $t = 0.01$, $w = 5$, speed up active. Happiness parameters are: $\alpha = \beta = 1$, $\gamma = \delta = 2$.

high variance in the resulting box plot. This variance decreases afterwards. Only UCB 1 has a higher variance once, at time step 350. The corresponding average platoon size has a similar development. The average platoon size increases to approximately 1.8 until time step 100. Afterwards, the average platoon size stays in the interval $[1.8, 2.4]$. The ε-greedy algorithm is also slightly, but not significantly behind the other algorithms at second 100. Afterwards it catches up. The decrease takes place at time step 350. After second 450 a slight improvement occurs. No algorithm is significantly better than the others, neither in happiness nor platoon size.

Figure 4.8 shows a simulation with weighting parameter setup $\alpha = \beta = 1$ and $\gamma = \delta = 2$. Speed deviation and platoon size are double weighted. This is the standard parameter setup for all simulations of this work. In comparison to figure 4.7 the average happiness is similar. All results have decreased about 0.02 in happiness. Starting at time step 100, the happiness is in the interval $[0.69, 0.76]$. At second 350 a decrease starts, which lasts until second 400. Afterwards there is a slight improvement. After 100 seconds, the variance of all algorithms decreases. Especially UCB 1 has a high decrease of variance. No algorithm is significantly better than the others. The average platoon size also behaves similar to figure 4.7. In the first 100 time steps, the only difference is, that ε-greedy is now on the same level as the other algorithms. After second 100, the platoon size is in the interval $[1.9, 2.5]$. Again, a decrease starting at second 350 and ending at second 400 takes place. Afterwards the average platoon size slightly improves. In the end, Thompson-Sampling and ε-greedy are slightly better than Heinovski.

In these simulations, we changed two weighting parameters at the same time and made some changes. To get a better understanding, we repeat the scenario with only one factor weighting double. Again the scenario from figure 4.7 is the baseline. Figure 4.9 shows a simulation with weighting parameter setup $\alpha = \beta = \delta = 1$ and $\gamma = 2$. The speed deviation is double weighted. The overall happiness improvement is about 0.04. Immediately at the start, the happiness improves about 0.05. Afterwards the happiness develops similarly to the other scenarios again. From second 100 until the end, the happiness is in the interval $[0.73, 0.81]$. The variance of all algorithms is small in comparison to other scenarios. Again from time step 350 to 400, there is a decrease. Afterwards the average happiness increases slightly. The

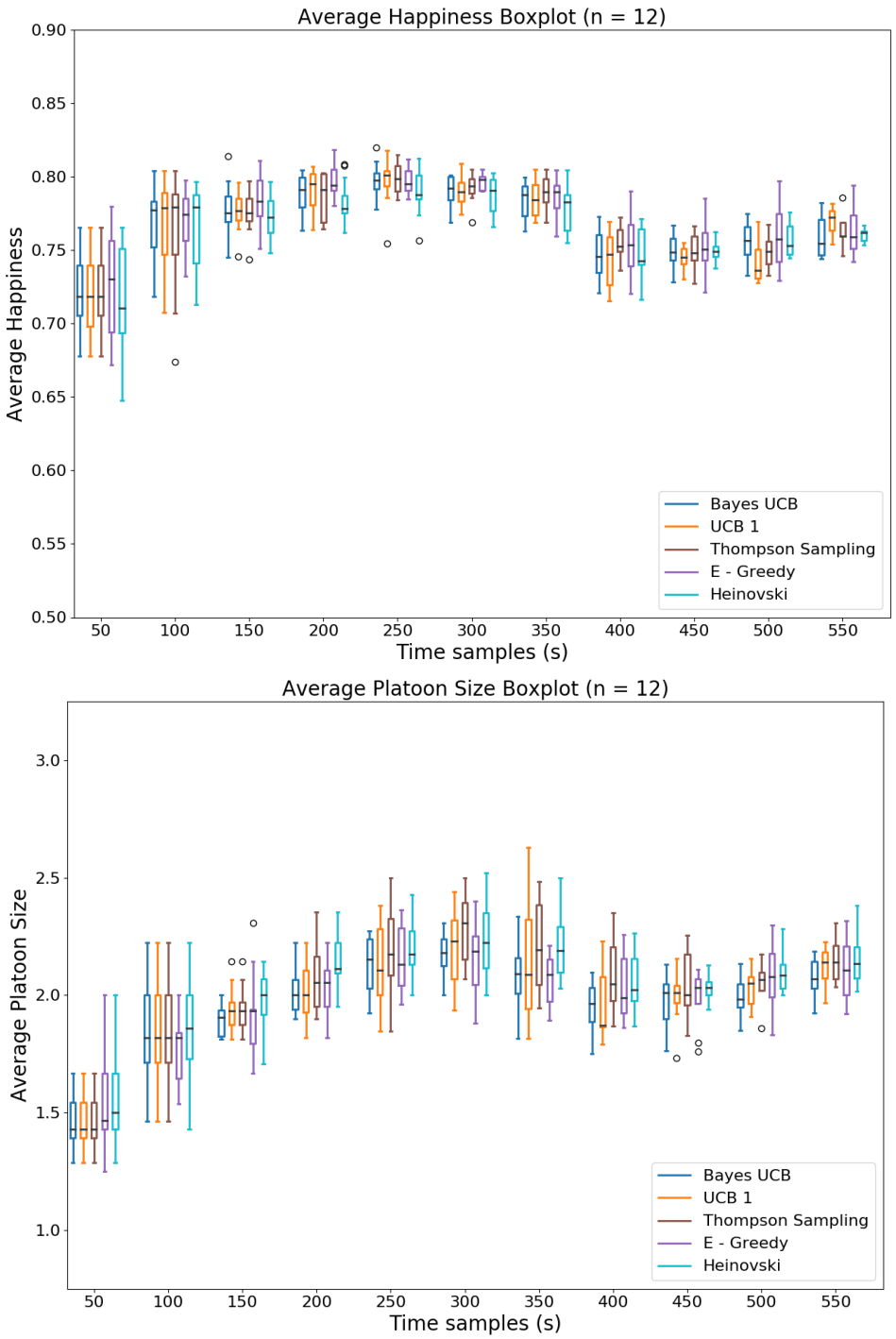

Figure 4.9: Scenario B from table 4.1. Average happiness (upper plot) in comparison to the average platoon size (lower plot). *Dynamic scenario.* $t = 0.01$, $w = 5$, speed up active. Happiness parameters are: $\alpha = \beta = \delta = 1$, $\gamma = 2$.

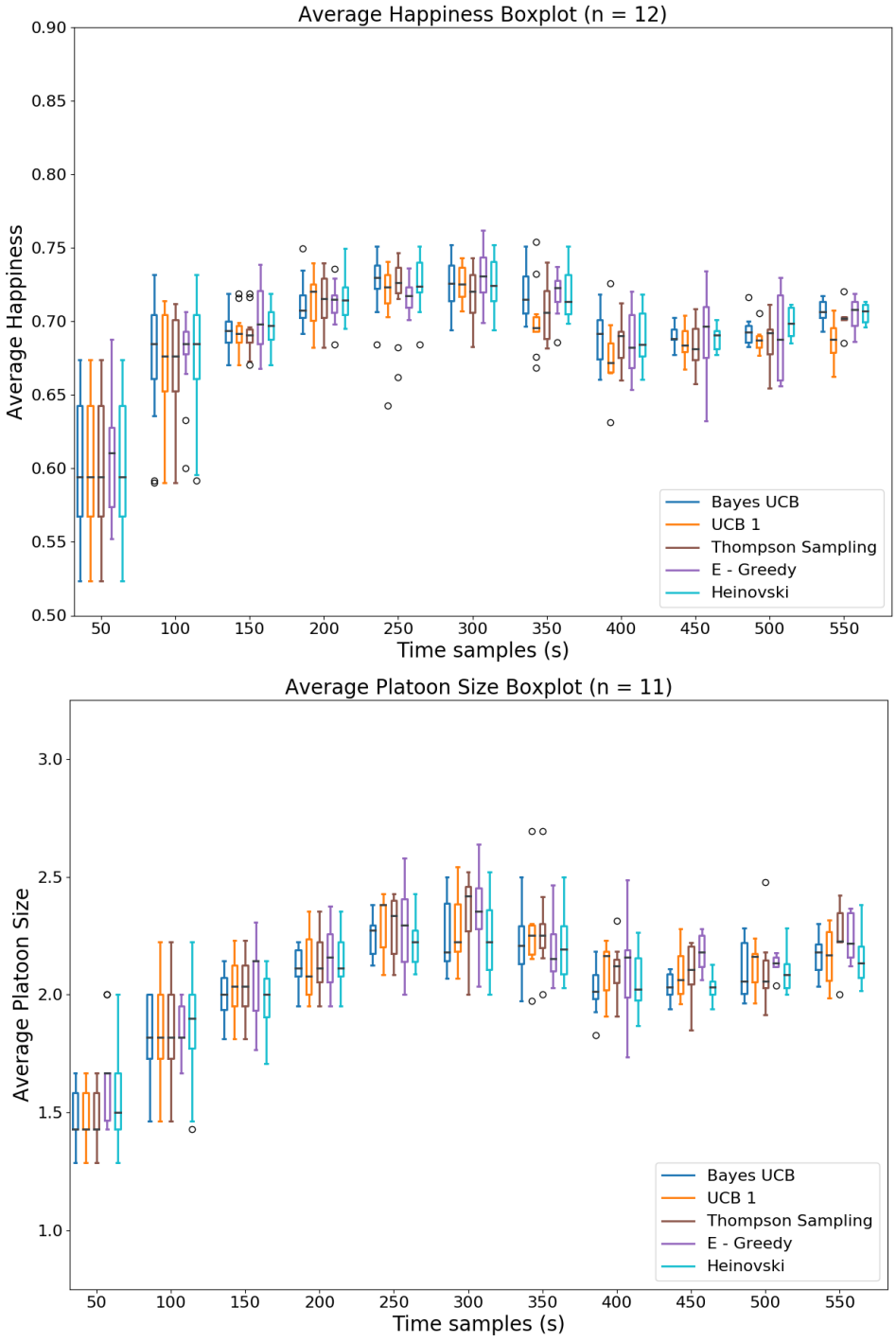

Figure 4.10: Scenario A from table 4.1. Average happiness (upper plot) in comparison to the average platoon size (lower plot). *Dynamic scenario.* $t = 0.01$, $w = 5$, speed up active. Happiness parameters are: $\alpha = \beta = \gamma = 1$, $\delta = 2$.

average platoon size is about 0.1 below the baseline. In difference, ε-greedy is not worse than the other algorithms at second 100. Therefore it is slightly worse than UCB 1, Thompson-Sampling and Heinovski afterwards. The biggest difference between the algorithms is at second 350, where a decrease starts again. After that decreasing phase the platoon size of all algorithms is on the same level in the increasing phase beginning at time step 450. At the end all algorithms except Bayes UCB are on the same level. Bayes UCB is close to it and not significantly worse.

Last, figure 4.10 shows a simulation with weighting parameter setup $\alpha = \beta = \gamma = 1$ and $\delta = 2$. The platoon size is double weighted. The starting happiness decreases about 0.06 in comparison to the baseline. The behavior is again similar. The average happiness stays in the interval $[0.72, 0.79]$ from second 150 until the end. The variance of UCB 1 decreases, while the variance of ε-greedy increases. At second 550 UCB 1 is significantly worse than the other algorithms. Although Thompson-Sampling has a small variance, the data of that box plot is based on six non-aborting simulations. The average platoon size, until 100 seconds, behaves similarly. Afterwards the platoon size is in the interval $[1.9, 2.5]$. ε-greedy has a higher variance. At time step 550, Thompson-Sampling is significantly better than Heinovski, Bayes UCB and UCB 1. Only ε-greedy can keep up.

4.3 Expected Problems

Section 3.5 mentioned expected problems in system stability. Due to computational cause, we have a limited amount of vehicles in the system. We will research the settling time and existing equilibrium in section 4.3.1. Another mentioned problem is the disruption of platoon forming due to dynamic switching. A platoon change may lead to a happiness decrease inside the joined platoon. Furthermore, this may result in another switching. The system will oscillate. Section 4.3.2 deals with this problematic.

4.3.1 System Stability

All plots in this section are from simulations with the same parameters. The upper plot always shows a *start to end scenario* (Scenario G from table 4.1) and the

lower plot a *dynamic scenario* (Scenario E from table 4.1). The *decision threshold* is set to $w = 5$. The situation was tested with *platoon switching threshold* $t \in [0, 0.01, 0.02, 0.04]$. The observations are similar. For this visualization we chose $t = 0.04$ and the UCB 1 algorithm because this setting has the biggest number of time steps simulated before aborting in both the *start to end scenario* as well as in the *dynamic scenario* [1].

Figure 4.11 shows the amount of cars in the system separated into states. Until second 180 the graphs look similar. Afterwards from second 180 to 300 the amount of *no platooning*, *joining* and *single cars* increases slightly in the *dynamic scenario*. It takes approximately 170 seconds for the fastest cars to arrive at exit one. Some cars leave their platoons, new cars are appearing on the highway and are looking for a platoon to join. After second 300, the amount of *platooning cars* increasing rate in the *dynamic scenario* has peaks at seconds 310, 410 and 480. During the *start to end scenario* the increase of the amount of *platooning cars* is smoother. Until the end the system has not reached the maximum amount of vehicles. In contrast, in the *dynamic scenario* the maximal amount is reached at second 500. This is caused by the appearing rate of new cars. While in the *start to end scenario* all cars have to appear in the beginning, the *dynamic scenario* has more entries to get vehicles into the system. The amount of cars in other states is higher in the *dynamic scenario*, but remains on a low level. Both simulations are similar in the end. The amount of platooned vehicles can not give any information about the average happiness or platoon size.

In figure 4.12 the average platoon size is plotted for both scenarios. While the amount of cars in specific states is similar, there are major differences in the average platoon size. The upper plot shows the *start to end scenario*. First, the speed up leads to a fast initial platooning. The average size of a platoon increases to 2, which means, that every car has joined its neighbor for a first platoon. Afterwards, the decision algorithms start working. Heinovski has a minor improvement because only single cars can join a platoon. Already joined cars are never able to switch again. After second 250 the average platoon size remains on the same level of 2.4 cars. The

[1] This simulation is based on the random seed 11 for the pseudo random number generator

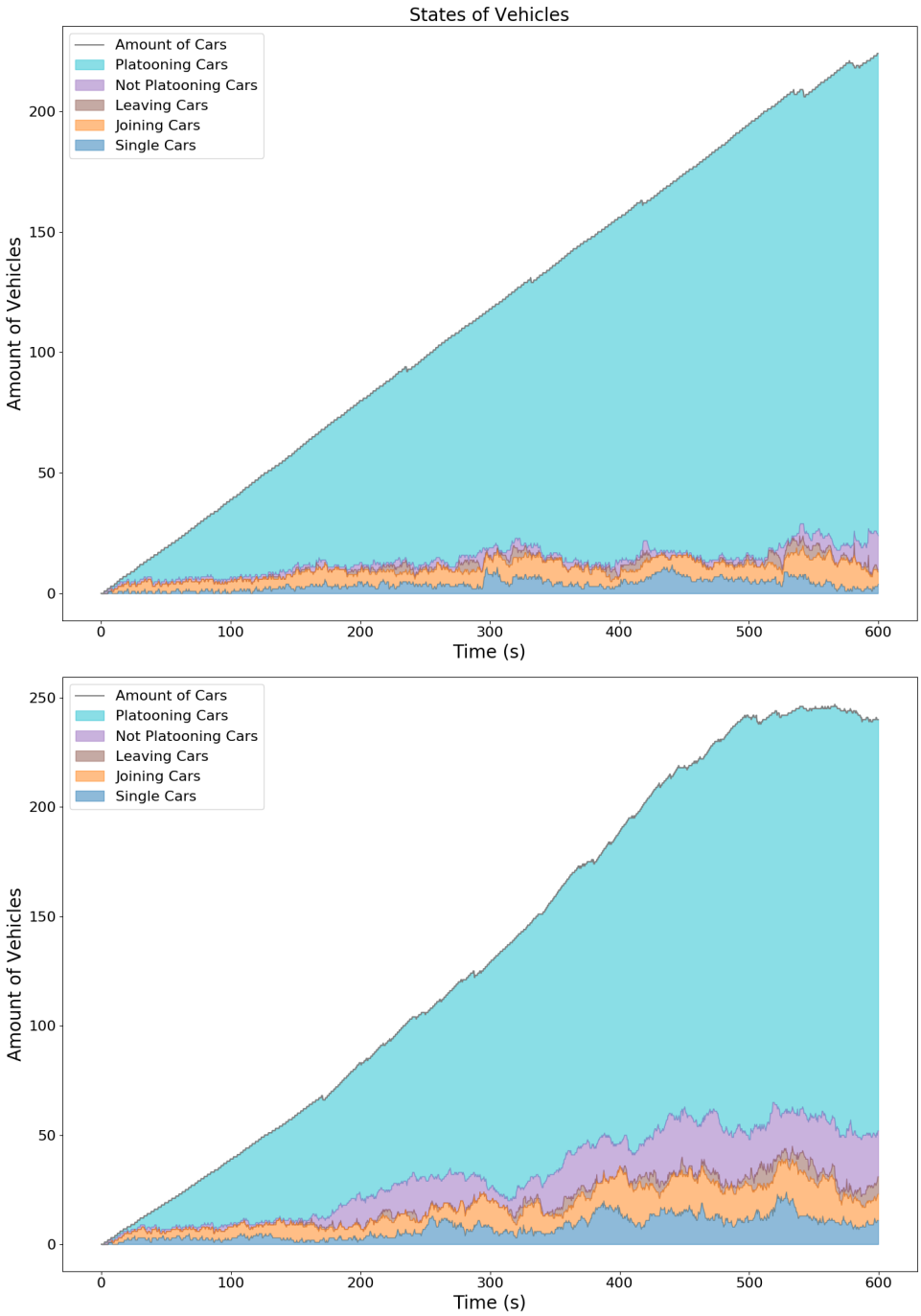

Figure 4.11: Scenario G (upper) and E (below) from table 4.1. Amount of vehicles in the system distinguished by state. The legend shows the separation into different states. The maximum number of vehicles allowed is 250. The upper plot displays a *start to end scenario*, the lower one is a *dynamic scenario*. $w = 5$, $t = 0.04$, speed up active, UCB 1, Seed 11.

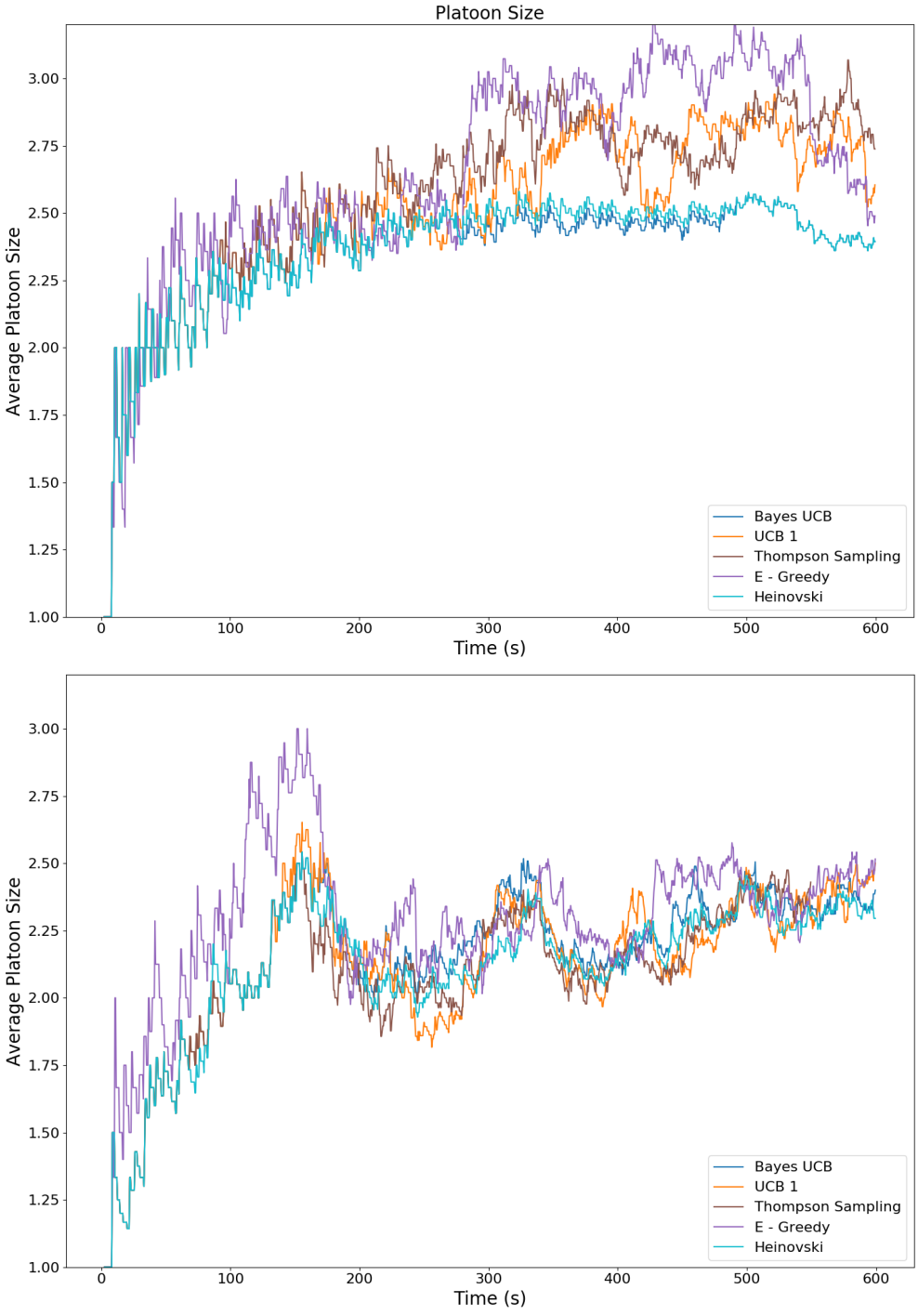

Figure 4.12: Scenario G (upper) and E (below) from table 4.1. Average platoon size of a single simulation. The lines symbolize different algorithms. Upper plot shows a *start to end scenario*, the lower one a *dynamic scenario*. $w = 5$, $t = 0.04$, speed up active, UCB 1, Seed 11.

Bayes UCB behaves similarly. The simulation run with that algorithm crashes at second 500. Until then, the average platoon size follows the one of Heinovski's algorithm. Starting from second 300, ε-greedy, UCB 1 and Thompson-Sampling have a better average platoon size, where ε-greedy outperforms the other two algorithms. At second 550, the first vehicles are leaving the highway. Single cars will appear at the start to compensate the leaving cars. This results in a slow decrease of the platoon size on all algorithms.

In comparison the *dynamic scenario* in the lower plot has some oscillations until the end of the simulations. The first platoon size decreasing situation occurs at second 180. Afterwards every 180 seconds another decreasing situation follows. This fits the average time a vehicle needs to drive from one exit to another. The ε-greedy algorithm performs best in getting a high average platoon size, but also has one of the biggest decreases afterwards. UCB 1 and Thompson-Sampling have a similar but flatter curve. There is no significant difference, between both of these algorithms. Again Bayes UCB and Heinovski behave similarly. There is also no significant difference between these two algorithms. Due to vehicles joining and leaving the highways later on, Heinovski's approach has some average platoon size changes in between. In comparison to the other algorithms, the change of average platoon size remains on a small level. Only single cars can join a platoon and only vehicles leaving the highway will leave a platoons in this approach. In the end there is a minor difference between the algorithms. Bayes UCB and Heinovski have an average platoon size of 2.3 cars. ε-greedy and UCB 1 have an average platoon size of 2.4 cars. Thompson-Sampling stops at second 550. At that time it behaves more like UCB 1 and ε-greedy algorithms.

Last, we take a look at the average happiness of both simulations. The happiness is separated into four happinesses components. Each is weighted in order to get the happiness plotted in this diagram. The weighting is:

$w_{\text{speed deviation}} = 2$, $w_{\text{distance to end}} = 1$, $w_{\text{distance in between}} = 1$ and $w_{\text{platoon size}} = 2$.

In figure 4.13 the happiness of the *start to end scenario* (upper diagram) is plotted against the happiness of the *dynamic scenario* (lower diagram). We start again

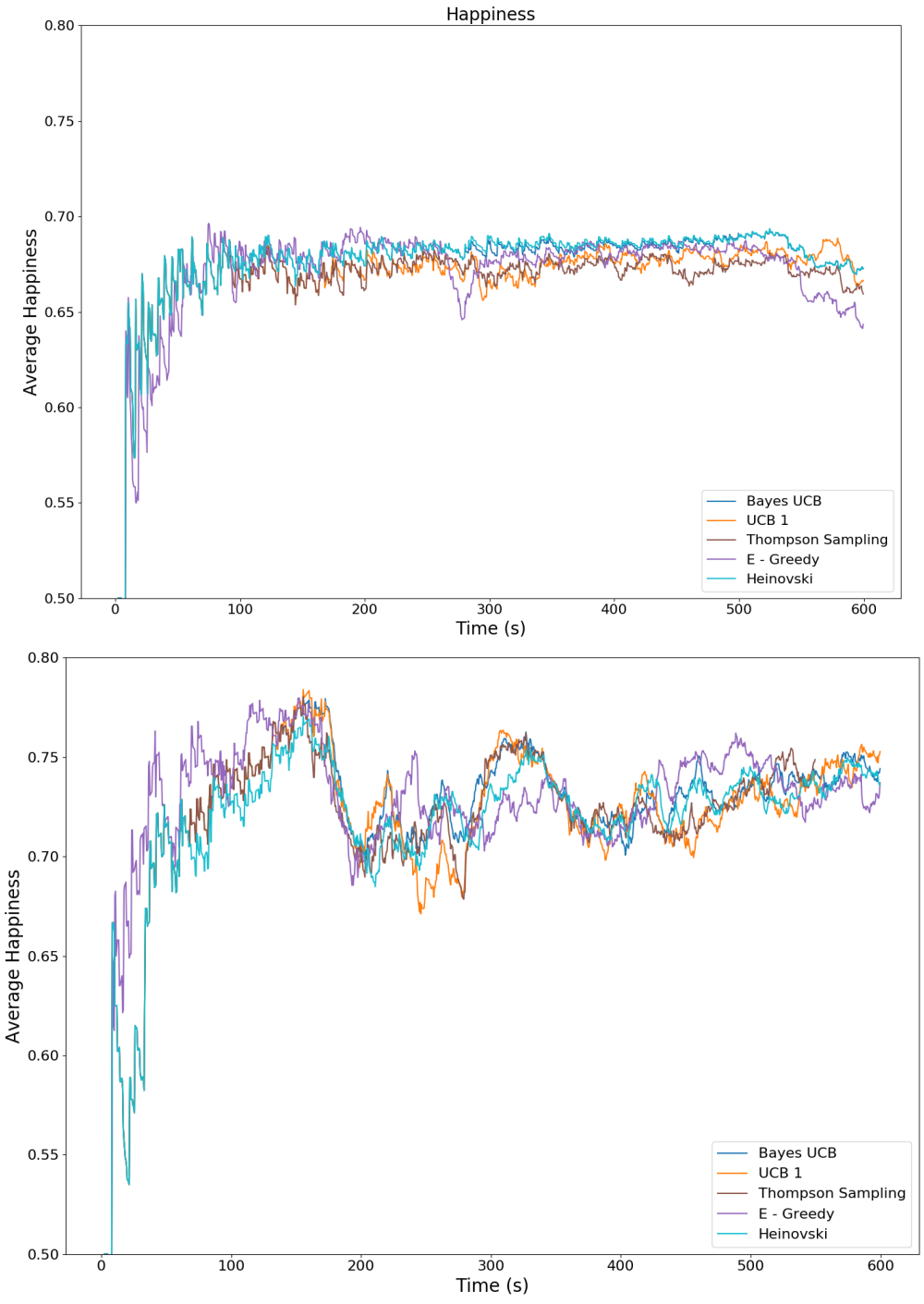

Figure 4.13: Scenario G (upper) and E (below) from table 4.1. Average happiness of a single simulation. The lines symbolize different algorithms. Upper plot is a *start to end scenario*, the lower one is a *dynamic scenario*. $w = 5$, $t = 0.04$, speed up active, UCB 1, Seed 11.

with the description of the upper diagram. With the first platoon formation at the start, the happiness across all algorithms grows to a level of about 0.67. Here, all algorithms remain stable from second 100 to 550. In the end, the average happiness is inverted to the average platoon size. While ε-greedy, UCB 1 and Thompson-Sampling have a higher average platoon size, the happiness of them is most of the time below the happiness of Bayes UCB and Heinovski. The graphs of ε-greedy, UCB 1 and Thompson-Sampling seem to start decreasing in the end. This goes hand in hand with the first vehicles leaving the highway, compensated by single cars with lower happiness.

There is an oscillation in the *dynamic scenario* we have already seen in the diagram of the average platoon size (figure 4.12). Besides this oscillation all algorithms have a similar progress. ε-greedy and UCB 1 algorithms have more peaks. After the peaks, their happinesses adjust to the happinesses of the other algorithms again. There is no inverted average happiness in relation to the average platoon size in the end. While the happiness of ε-greedy is below the happiness of Bayes UCB and Heinovski, the happiness of UCB 1 remains higher. Thompson-Sampling aborts at second 550. At that time this algorithm behaves similarly to UCB 1.

4.3.2 Disruption of Platoon Forming due to Dynamic Switching

Each member of a platoon scans its neighborhood continuously in order to find better fitting candidates for platooning. If the algorithm decides, that there is a candidate with higher happiness, the platoon changing process starts. The success of a changing maneuver is not guaranteed. However, this process may abort because of external circumstances, like other vehicles accidentally blocking the maneuver. A successful changing maneuver should lead to an improvement of happiness of the changing car. A positive happiness change at least of the joined platoon is not expected because there is no acknowledgment necessary allowing a vehicle to join a platoon. If the summed up happiness of all members decreases, a platoon change may force another platoon change. If we take a look at a scenario, where two similar platoons are driving next to each other, such a behavior may lead to constantly recurring platoon changes.

Figure 4.14 shows histograms presenting the dynamics of platoon changing processes in two different scenarios. The first and third histogram show the happiness improvement of a joining vehicle. The second and fourth histogram show the happiness improvement of the platoon joined by that vehicle. The route is a *dynamic scenario* with activated speed up. The *decision threshold* is set to $w = 5$. The upper two diagrams are plotted with *platoon switching threshold* $t = 0$. The histograms below are plotted with $t = 0.04$. The scenario is simulated with $t \in [0, 0.01, 0.02, 0.04]$.

The improvement of happiness of the joining car behaves similarly in both simulations. UCB 1, Thompson-Sampling and ε-greedy have two peaks at 0 and at 0.38. With $t = 0$ Bayes UCB has the same peaks. With $t = 0.04$, peaks are at 0.1 and 0.35. Looking at the happiness improvement of the platoon, UCB 1, Thompson-Sampling and ε-greedy have the same peaks at 0 and -0.32. Also here, Bayes UCB has the same peaks with parameter $t = 0$. There is only one peak at 0 for parameter $t = 0.04$. As $t = 0.04$ is the more conservative scenario, the frequency of platoon changes decreases over all algorithms. Thompson-Sampling decreases most, while Bayes UCB is nearly unaffected. The frequency of ε-greedy and UCB 1 decreases at the same level.

The improvement of happiness due to platoon changes is always positive, except for the ε-greedy algorithm with $t = 0.04$. First, we exclude Bayes UCB from the analysis. Thompson-Sampling performs best. A platoon switch gains the switching car an average improvement of 0.17 or 0.16 happiness (dependent on the scenario), while the sum of happiness of all other members decreases only by 0.07. ε-greedy performs worst. The first scenario results in 0.11 happiness improvement for the single car and -0.08 for the platoon, thus means there is an average per improvement of 0.03 per change. Meanwhile with $t = 0.04$ ε-greedy has an average per improvement over all vehicles of -0.06 per change. Bayes UCB has an ambivalent behavior. It performs worst with $t = 0$, but performs best with $t = 0.04$. In comparison to the other algorithms, the amount of changes is far lower. Thus the best average platoon changing is neglected by the small amount of changes, if we look at the overall happiness improvement.

The scenarios P with $t = 0.01$ and Q with $t = 0.02$ of table 4.1 are in the ap-

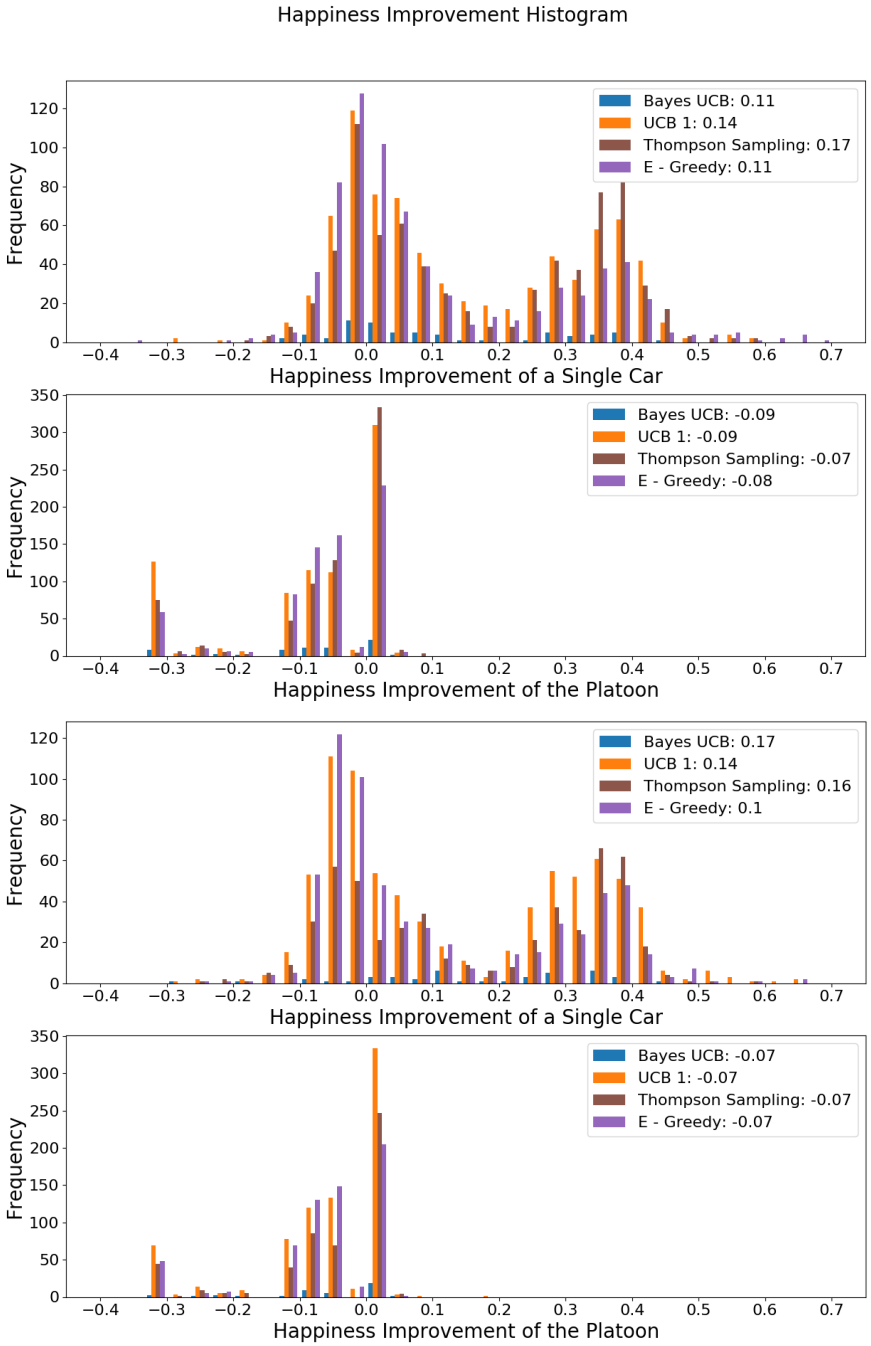

Figure 4.14: Scenario D (upper) and E (below) from table 4.1. Happiness improvement due to platoon changing maneuvers. Plot one and three show this improvement for the joining vehicle. Plot two and four show the sum of improvement of all members from a platoon, after a new car has joined it. The number in the legend is the average improvement per change.

pendix. The scenario with $t = 0.01$ behaves similarly to parameter $t = 0$ and the scenario with $t = 0.02$ behaves similarly to parameter $t = 0.04$. Also with parameter $t = 0.02$, ε-greedy is the only algorithm with a negative improvement of -0.06 per change.

4.4 Comparison of Multiple Scenarios

In the following we compare the average happiness and average platoon size of multiple scenarios, shown in figure 4.15. We have used these scenarios in the previous sections. In comparison to the before mentioned box plots, we do not use a distribution over time anymore. There are five box plots for every scenario, one for each algorithm. The box plots are generated from the average happiness / platoon size of an algorithm over 12 seeds. In every seed, the data from time step 150 until the end is used. This results in a maximum of 120 data points for each box plot, if no simulation aborts. The scenarios are labeled with letters. The parameter setup of each letter is listed in table 4.1.

We separated the algorithms into groups. The group with the scenarios A, B and C is from section 4.2.3, where different weighting factors were tested. Scenarios D to G are from section 4.2. We simulated a *dynamic scenario* (scenarios D and E) and a *start to end scenario* with different *platoon switching thresholds t* in that section. To get an idea of the influence of the speed up, the scenarios H to K are nearly identical to the scenarios D to G. The difference is a deactivated speed up. Scenarios H and I are in a *start to end scenario* and scenarios J and K are in a *dynamic scenario*.

We test the result with the Wilcoxon rank-sum test. We do a t-test with the null hypothesis, that the data distribution of two algorithms is the same. Among a scenario all algorithms are compared with each other. All algorithms have different distributed data with certainty $p < 0.05$, except the pairs shown in table 4.3.

Scenario B has the highest average happiness. It is followed by scenarios C, D and E. Scenarios D and E are about the same level. All *start to end scenarios* (F, G, H and I) have about the same average happiness level, too. The scenarios J and K perform the worst. These are *dynamic scenarios* without speed up. In scenarios

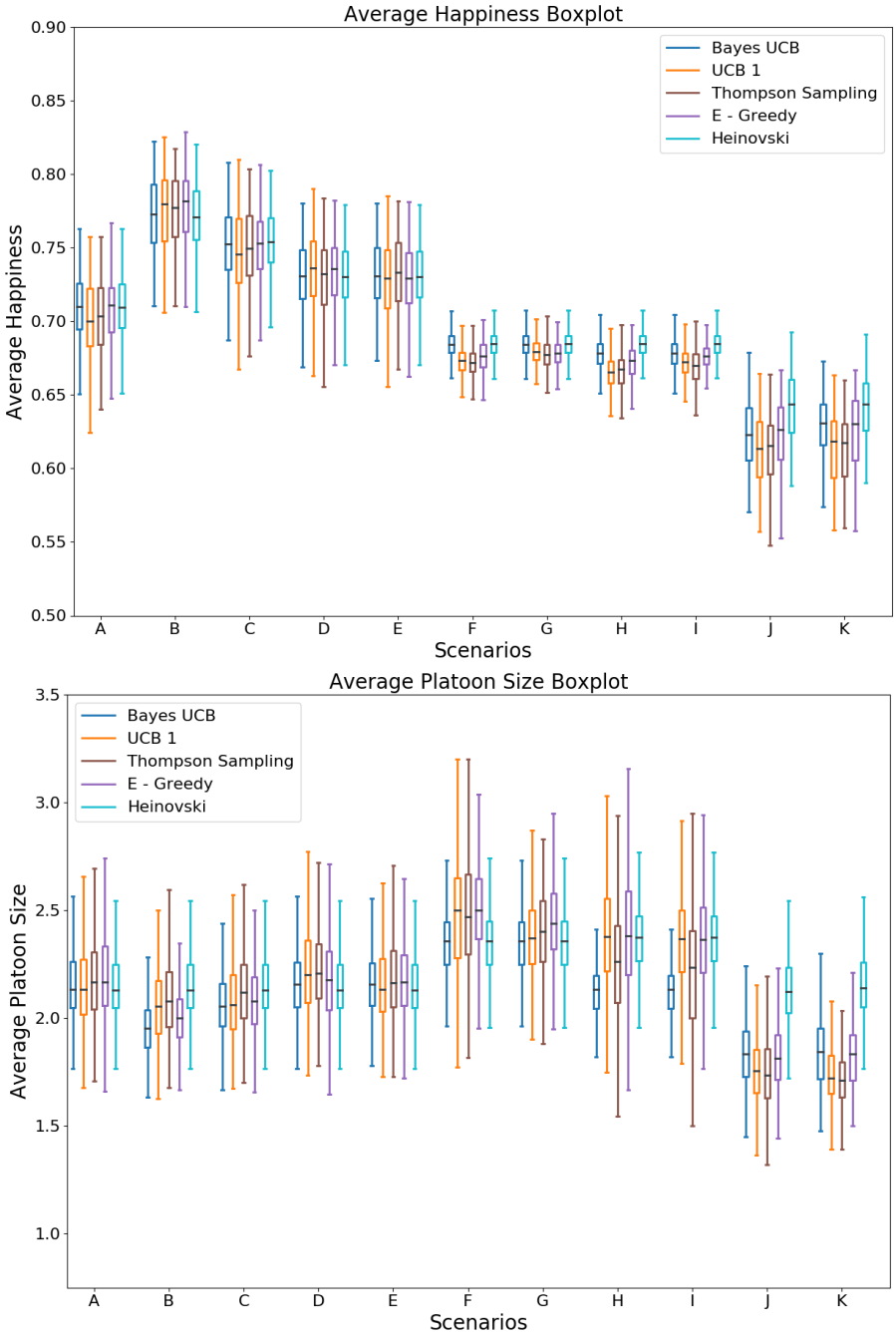

Figure 4.15: Average happiness (upper plot) in comparison to the average platoon size (lower plot) of multiple scenarios. Specific parameters of each simulation are listed in table 4.1.

Scenario	algorithms tested	p
A	Heinovski / Bayes UCB	0.15
A	Heinovski / UCB 1	0.49
D	Heinovski / Bayes UCB	0.07
D	UCB 1 / Thompson-Sampling	0.34
E	UCB 1 / Bayes UCB	0.71
E	UCB 1 / Heinovski	0.09
E	Thompson-Sampling / ε-greedy	0.45
F	Heinovski / Bayes UCB	0.60
F	UCB 1 / Thompson-Sampling	0.42
G	Heinovski / Bayes UCB	0.80
I	Heinovski / ε-greedy	0.82

Table 4.3: Wilcoxon rank-sum test results for all pairs of algorithms with a similarity of their data distribution with certainty $p > 0.05$. Not published pairs of algorithms have a certainty of $p < 0.05$.

J and K Heinovski has a higher average happiness than the other algorithms. These are the only scenarios, where an algorithm is significantly better than the comparing ones.

All *start to end scenarios* (F, G, H and I) have the highest average platoon size. With deactivated speed up (scenarios H and I) the Bayes UCB algorithm performs worse than the other algorithms. With activated speed up (scenarios F and G), the algorithm performance is dependent on the *platoon switching threshold* t. In scenario F this parameter is set to $t = 0$. Thompson-Sampling, UCB 1 and ε-greedy algorithms are on the same level. These algorithms perform significantly better than Bayes UCB and Heinovski. Bayes UCB and Heinovski are on the same level as well. In scenario G, $t = 0.04$ is set. With this parameter no algorithm is significantly better than the others. Among the dynamic scenarios with activated speed up (A, B, C, D and E) there is no algorithm significantly better than the others.

5 Discussion

Finally, we want to interpret our results. A basic understanding of our parameters is essential to get an idea of how the simulation behaves. Section 5.1 is an analysis of the parameters *scenario type*, *decision threshold*, *platoon switching threshold* and the *speed up*. Another essential for further analysis is a perfectly working simulation. We are facing some technical challenges with our simulation and analyze them in section 5.2. A simulation has a settling time as well as a decay time. In the time between, the platooning influences are observed with minimal external disruption. We show in section 5.3, that these influences are dependent on the scenario. *Start to end scenarios* are better to observe the influences than *dynamic scenarios*. In section 5.4 the topic is about dynamic platoon switches without benefit. Vehicles changing platoons may lead to problems. One of the expected problems was a repeatedly dynamic switching behavior among two platoons driving next to each other. Cars are changing from one platoon to the other, thus leading to a cycle, where platoon changes lead to no benefit. Finally, section 5.5 is an overall summary of the behavior of machine learning algorithms implemented to solve this multi-armed bandit problem. We handle several aspects and get to a final conclusion of how well our approach is working.

5.1 Influence of Parameters

We introduced several parameters. By tuning them, the scenario varies a lot. Parameters we chose for tuning are the *type of scenario*, the *decision threshold* w, the *platoon changing factor* t and *speed up* activation. In the following, we take a look at each parameter and describe the influence on the whole system.

Type of scenario: Among the simulations, we used a *start to end* and a *dynamic scenario*. We refer to figure 4.15 and the corresponding parameter table 4.1.

Pairs, where the only parameter difference is the chosen scenario, are (D, F), (E, G), (J, H) and (K, I). First mentioned is a *dynamic scenario* and second mentioned is a *start to end scenario*. Among all pairs the average platoon size is approximately 0.2 vehicles higher in the *start to end scenarios*. With deactivated speed up ((J, H) and (K, I)) this difference raises to approximately 0.5 vehicles. This matches the expectations that a platoon in a *start to end scenario* is not disrupted by vehicles leaving or entering the highway, thus the algorithms can continue optimizing. Vehicles are leaving and entering the highway at 6 kilometer intervals in a *dynamic scenario*. As soon as a platoon has formed and improved its constellation, platoon members will leave and new sub-optimal fitting vehicles will join at highway entries. This leads to unsettled phases. Afterwards multiple platoons have to reorganize themselves. These dynamics are missing in a *start to end scenario*. Vehicles have about 22 kilometer of highway length to organize in well-fitting platoons without any disturbances. A *dynamic scenario* leads to more platoon changes and a reduced average platoon size. The effect on the happiness remains unclear. With speed up activated, the *dynamic scenarios* of pairs (D, F) and (E, G) have higher happiness. With speed up deactivated, it is the other way round. We had noticed, that a higher average platoon size correlates with a lower average happiness. The *speed up* helps the *dynamic scenario* to form platoons fast. Due to disruptions at highway entries and exits in the *dynamic scenario* the platoon forming process may be too slow, thus increasing the average happiness.

Decision threshold w: We refer to figure 4.4. The only difference is the decision threshold $w = 10$ in the upper and $w = 5$ in the lower plot. As the approach of Heinovski and Dressler [14] is not affected by this parameter, this algorithm performs great as a baseline. With $w = 10$ no algorithm performs better than Heinovski. We see a steadily but slowly increasing platoon size of all other algorithms, except Bayes UCB. With $w = 5$ the behavior is the same, but the platoon size increases faster. In the end, the ε-greedy algorithm outperforms Heinovski significantly. The decision threshold is the amount of how many times a vehicle must explore a candidate, before the candidate can be chosen as a best neighbor. A lower value leads to a faster decision. This leads to a faster increase of the average platoon size. If the value w is too low, there are negative effects. For example, a value of $w = 1$ means, that a vehicle for platooning is chosen directly after the first exploration. Vehicles

match with the first car in sensor range. But this is no desirable behavior because it replaces the algorithm with a fast-greedy approach. Further testing was not done. One stand-alone simulations with $w = 1$ is shown in the appendix. We generalize, that a lower decision threshold leads to a faster decision, thus increasing the growth of the average platoon size until a certain threshold value, where the average platoon size decreases again.

Platoon changing threshold t: We will focus on the plots of section 4.2.1 and section 4.2.2. We simulate scenarios using the parameter set $t \in [0, 0.01, 0.02, 0.04]$. The two plots, with the biggest differences are printed. These are the scenarios with $t = 0$ and $t = 0.04$. We expected this behavior to increase, if using a small threshold, and decrease the average platoon size with a high threshold. In the beginning of the *start to end scenario*, there is no difference between both simulations. Most platoons are heterogeneous. This leads to a high happiness improvement by joining another platoon, thus leading to an improvement of the average platoon size. Whether a threshold is set to 0 or 0.04 does not make any difference because most of the improvements are higher than the threshold. The platoons get more homogeneous towards the end. There are only small happiness improvements possible. A threshold, that a new platoon must be, for example, at least $t = 0.04$ happiness points better, restricts most vehicles from changing to another platoon. Only a few changes take place. The average platoon size stagnates. The scenario with a lower threshold improves faster. When looking at the *dynamic scenario*, there is no significant difference between these simulations. When passing a highway exit, most platoons will adapt, thus loosing members and getting new ones. The platoon mixes up and can hardly improve. As mentioned before, while being a heterogeneous platoon, vehicle changes bring a high happiness improvement. After some time the platoons are on a homogeneous level. Platoon changes only lead to small improvements. This situation hardly occurs in the *dynamic scenario* because the platoons are mixed at the highway exit before. This happens every 6th kilometer. As a result the difference among different platoon change thresholds is not significant. Generally spoken, A high platoon switching threshold leads to a more conservative scenario with fewer platoon changes.

Speed up activation: On the basis of the implemented happiness model, we have the following assumption. Driving in a platoon is always better for a vehicle than

driving on its own. We compare figure 4.4 and figure 4.6. We assume, that the first decision for platooning has great impact on the scenario. Afterwards platoon switches only have a minor affect on the scenario. Without activated speed up, the algorithm of Heinovski outperforms the others in the beginning. It lasts until the end of the simulation, when the other algorithms get on the same level as Heinovski. In order to compensate this advantage of fast decisions of Heinovski, an activated speed up leads to immediately forming a platoon, when a single car detects another single vehicle. As a result, all other algorithms are on the same level in the beginning. Afterwards, the decision process of the different algorithms leads to different results. UCB 1, ε-greedy and Thompson-Sampling are significantly better in the end due to this speed up. The amount of platoon switches is not different between a scenario with *activated speed up* and a deactivated one. Small differences result from having more time to switch because the first platoon formation was faster. All in all, our assumption seems to be correct. The *speed up* is essential for a better comparability of the algorithms among our simulation time of 600 seconds.

5.2 Implementation and Simulation

A major problem are crashing vehicles, that also crash the simulation sometimes. The analysis in section 4.1 is based on eight simulations in the parameter set $t \in [0, 0.01, 0.02, 0.04]$ simulated in a *start to end* and *dynamic scenario*. The platoon switching threshold t and the type of scenario are parameters with high influence on the amount of platoon changes. As mentioned in section 5.1 the platoon switching threshold behaves anti proportional to the amount of platoon changes. A simulation with $t = 0$ has more changes than a simulation with $t = 0.04$. A similar effect on the system has the selected platooning scenario. A *start to end scenario* always has fewer changes than a *dynamic scenario*. Table 5.1 lists the algorithms with scenario, amount of platoon changes, amount of crashes and average simulation steps. The table is sorted by average simulation steps.

We can generalize, that a high amount of platoon changes leads to a higher amount of vehicles crashing. This leads to a lower average amount of simulation steps. A scenario with a lot of vehicle crashes has a high probability to abort. There seems to be relation between the number of platoon changes an the amount of crashes.

Algorithm	**Dynamic (D)** or **Start to End (SE)**	**platoon changes**	**vehicle crashes**	**simulation steps**
Bayes UCB	SE	0.5	0.0	600
Heinovski	SE	0.0	0.0	600
T. S.	SE	60.9	6.1	584.9
ε-greedy	SE	59.1	3.7	582.3
Heinovski	D	0.0	11.8	540.3
Bayes UCB	D	9.9	11.3	512.9
UCB 1	SE	64.1	7.1	508.2
UCB 1	D	117.8	20.2	480.6
T. S.	D	90.6	21.0	479.2
ε-greedy	D	82.1	17.9	462.4

Table 5.1: Overview about the scenarios and algorithms. For each entry, the average amount of platoon changes, the average amount of vehicle crashes and the average amount of simulation steps is listed. Table is generated from data of the visualization in section 4.1. Thompson-Sampling is abbreviated as T. S.

UCB 1, Thompson-Sampling and ε-greedy have a lot of changes. Bayes UCB only has a small amount of changes and Heinovski does not have any platoon changes. As a result UCB 1, Thompson-Sampling and ε-greedy more crashes than Bayes UCB and Heinovski. A *start to end scenario* has less platoon changes than a *dynamic scenario*. This correlations with the amount of simulation aborts as well. There are a lot of vehicles entering or leaving the highway in a *dynamic scenario*. When leaving the highway, a car immediately quit its platoon. Cars, that enter the highway, immediately form a platoon. A high amount of simultaneously platoon switches, leaves and formations at a small stage of the highway is the consequence. This does not occur in a *start to end scenario*. A simulation error leading to an abort may happen sometimes or even never. A simulation may have a high amount of errors and runs until the end, while another simulation stops with the first error. This behavior is strictly dependent on the type of error.

5.3 System Stability

The analysis in section 4.3.1 leads to the conclusion, that the system is stable enough for observing platooning behavior. The settling time of about 450 seconds in a *dy-*

namic and more than 600 seconds in a *start to end scenario* does not affect the platoon formation much, despite the simulation time of 600 seconds as well as the limited amount of vehicles not being optimal. In order to establish longer ongoing simulations, we need to fix problems mentioned in section 5.1. Else we do not have a profound amount of data to rely on. Just under 40 percent of the amount of simulations are not aborting. At the current state of implementation we expect to have less than 25 percent of the simulations running until second 900. In order to get comparable scenarios, we need a parameter setup, where the same seeds tested with different algorithms have about the same time running before aborting. This is critical with about less than half completed simulation runs. If there is no limit of vehicles, the car density will grow too. A high density is an indicator for vehicles crashing, thus leading to new simulation crashes.

Furthermore, we can observe platooning behaviors better in a *start to end scenario* than in a *dynamic scenario*. There are external disturbances influencing the average platoon size as well as the average happiness. In an average platoon size box plot, the time step, where the highest vehicle density reaches an exit, is a turning point. However, we cannot estimate which part of increasing or decreasing platoon size is based on cars joining / leaving the highway and which part is based on the platooning algorithms. The *start to end scenario* is preferable as a test bed for a new algorithm. There, we can observe a specific behavior and assume, that this also has an effect on a *dynamic scenario*. In order to improve the algorithms, we should focus on *start to end scenarios*.

5.4 Disruption of Platoon Forming due to Dynamic Switching

By setting up a highly *dynamic scenario*, there was the question, if two platoons driving next to each other may lead to a constantly recurring platoon changing. If there are a lot of platoon switches and they do not lead to an overall improvement, this is a sign that such a circle of platoon changes may occur. We refer to the plots in section 4.3.2. We set up a happiness improvement histogram. There we can see the happiness improvement taking place by a platoon change. On the one

hand, we are looking at the happiness of the changing vehicle. This happiness should improve in most cases, since a better happiness was the reason to start this change. On the other hand a joining vehicle may reduce the happiness of the joined platoon. The happiness is a value in the interval $[0, 1]$, while a single vehicle has a happiness of 0.5. From observation we can assume, that most of the platooned vehicles have a happiness between 0.6 and 0.9. The histograms in figure 4.14 shows, that with UCB 1, Bayes UCB and Thompson-Sampling a switch generally improves the happiness of a single car more than 0.1, while the joined platoon sum of happiness decreases less than -0.1. We assume, that an overall happiness improvement of at least 0.02 per platoon changing maneuver (the addition of the single car and the platoon happiness improvement) should prevent the system from developing the cyclic behavior. All algorithms except ε-greedy improve about that amount. A ε-greedy approach with a high platoon switching threshold t leads to worse behavior. In that case, a platoon switch decreases the happiness about -0.06. In this way, we cannot estimate if ε-greedy leads in some scenarios to a cyclic behavior.

5.5 Comparison of Machine Learning Algorithms

The main objective of this thesis is researching algorithms from machine learning for adapting platooning as a multi-armed bandit problem. Among a variety of scenarios we have tested five different algorithms - ε-greedy, UCB 1, Bayes UCB and Thompson-Sampling as machine learning approaches and the approach of Heinovski and Dressler [14] as a baseline. A selection of tested scenarios is presented in figure 4.15. We will refer to this box plot.

As a main observation of the average platoon size, we hardly observe any difference between the algorithms among scenarios A to E. These are *dynamic scenarios*. The happiness differs among the scenarios. Also by looking at the happiness, there is a barely any difference among the algorithms. In the *start to end scenarios* F and G, the average platoon size differs among some algorithms. The platoon size with ε-greedy, Thompson Sampling and UCB 1 is slightly higher in scenario F. In scenario G, only ε-greedy is significantly better. The happiness of both scenarios has no significant difference among the algorithms. Finally, the scenarios H to K are the group of scenarios without an initial speed up. In the dynamic approach (scenario

J and K) the algorithm of Heinovski is significantly better in average happiness and average platoon size. The other algorithms are on about the same level. We perform a Wilcoxon rank-sum test in order to gain information about the similarity of the data distribution among the algorithms in the same scenario. The data basis is the average platoon size box plot comparison in figure 4.15. There are more similarities of data distributions among the *dynamic scenarios* (A - E) than among *start to end scenarios* (F - I). This is an indicator, that the algorithms behave more similar in the *dynamic scenarios*. If the system changes fast, a too high reaction time is a reason for this similarity. At the time of a decision, there may only one switching option be left, thus leading to the same decision of all algorithms.

All tested machine learning algorithms are not significantly performing better than the approach of Heinovski and Dressler [14]. The performance is highly dependent on the scenario. There are two main features for improving the machine learning algorithms: a faster decision leads to an improvement of the average platoon size. As we can see in scenarios with deactivated speed up, all approaches are significantly worse than Heinovski. In other scenarios, where we experimented with different decision thresholds w, a lower threshold leads to a faster improvement of the machine learning algorithms. Heinovski forms platoons fast. It outperforms the other algorithms at the beginning. The gap between those algorithms is closed over time. This leads to the second feature. The performance of machine learning algorithms increases in scenarios with less external changes. UCB, Thompson Sampling and ε-greedy are optimizing over time. If the scenario changes rapidly, the optimizing process is too slow to react. Another research about the decision thresholds w may help to improve that. Up to a certain point, a small threshold w speeds up the decision. We can observe this behavior in scenario D and F. D is a *dynamic scenario*. Before the algorithm can optimize to a certain level, platoons are disrupted because vehicles are leaving the highway. New vehicles are entering the highway as well. They choose the first available vehicle for initial platooning, before the algorithms optimize the platoon again. This initial platoon formation works well with single cars leading to a first platoon. But it has a negative effect on well-matched platoons and decreases the happiness of them, thus leading to platoon changes sooner or later. In scenario F, we do not have such external disruption. In this *start to end scenario* the algorithms have a total highway length of 22 kilometers to optimize the

platoons. The result is a steadily increasing average platoon size. After an initial increase the Heinovski approach levels out. There is no car left, that can join a platoon, because platooned cars are not able to leave their platoon. The amount of vehicles in platoon state is high, but this is done by a lot of small platoons. In all other algorithms, the average platoon size increases due to platoon changes of single vehicles. The homogeneity of platoons increases.

6 Conclusion

The objective of this thesis is to adapt algorithms from the machine learning sphere to platoon formation. Therefore, we model platooning as a multi-armed bandit problem. For a *start to end scenario*, the algorithms ε-greedy, UCB 1 and Thompson-Sampling are better than the baseline algorithm of Heinovski and Dressler [14]. For a *dynamic scenario* we have a negative result. Neither of the machine learning algorithms is significantly better than the baseline. The performance of the algorithms is highly dependent of the scenario setup. There is a variety of different parameters to be tested.

One of the main problems are crashes among vehicles, thus leading to simulation crashes. We did find setups to collect profound data in order to get to a conclusion. Nonetheless these simulation crashes disturb the research and make it difficult to analyze some behaviors. Simulation crashes are the results of vehicle crashes. An implemented routine detects crashed cars and removes them from the simulation. This routine prevents following vehicles from stopping, thus prevent the system to stop. There is a problem occurring in the communication with SUMO. In some cases, removed vehicles are still known by SUMO. This results in the program addressing vehicles, who are currently not part of the simulation anymore. A simulation crash is the consequence. The improvement of the simulation is essential to improve the quality of the results.

As mentioned above, each scenario is formed by a wide variety of parameters. We had to limit the selection of presented results and decide on which setups we want to focus on. We researched the speed of decision and tested the *decision threshold* $w = 5$ and $w = 10$. $w = 5$ results in faster decision making and better results in the average platoon size. We suggest a fast decision making as good, but a low threshold of $w = 1$ leads to every vehicle joining the first neighbor in range. This

behavior unifies the algorithms. We only test one scenario with $w = 1$ (shown in the appendix). This scenario has a high platoon switching rate as well as a high average platoon size. Therefore, the average simulation time is shortened, mainly because of the higher amount of platoon switches.

Further systematic testing is required in order to fit this parameter. An idea to be mentioned is a varying decision threshold w. Each vehicle has its own threshold. This value varies in dependence of the neighborhood density of the vehicle. A higher density (with more candidate to choose from) leads to a higher threshold w, while a low density leads to a small threshold. Another possibility we did not study is the frequency of algorithmic evaluation of the neighborhood. All simulations in this thesis are evaluating once per second. We did a simulation with an evaluation frequency of half a second, but the results decrease in average platoon size (This simulation is found in the appendix).

The happiness is adjustable by four sub-happinesses. These sub-happinesses are weighted differently. Our standard parameter setup is: $\alpha = 1$, $\beta = 1$, $\gamma = 2$ and $\delta = 2$. We tested other parameter setups as well with focus on the weighting factors of the speed deviation γ and of the platoon size δ. The average platoon size is influenced by the happiness. This thesis has not researched the behavior of the other sub happinesses *distance to end* and *distance in between two vehicles*. Besides the weighting factors, there may be other aspects to consider, that should have some influence on the behavior of the platoon formation. This is a field of research, where a wide variety of results is possible.

Last we did some research about the stability of the system. We came to the conclusion, that the current setup is sub-optimal, but works well for observation of platooning behavior. Interesting scenarios are a highway without a limit of vehicle and longer scenarios to arrange an equilibrium of vehicles. The appearing rate of vehicles is adaptable to a better fitting one. At the moment, vehicles are appearing every two to three seconds randomly at the start or a highway entry. Thereby vehicles are prevented from appearing at a specific highway entry before a predefined amount of time has passed (currently 170, 340 and 510 seconds). This behavior is chosen by experience and has no further scientific background. The aim is to get a high density of vehicles. A high density is required for platoon formation, where

a car can choose a candidate of several neighbors. In order to research these, it is prerequisite to avoid vehicle and simulation crashes.

There are two main aspects for further work. The needed time for decision making has a great influence on the observed behavior. One option is experimenting with the value of the decision threshold w, introducing a dynamic decision threshold (as mentioned above) and changing the frequency of neighborhood evaluation. Another option is to change the V2V-communication in an acknowledgment-based communication. Currently, a vehicle always decides to join a platoon on its own. There are situations, where the joining process leads to a disruption of the existing platoon and the benefits of the platoon are of no importance. With acknowledgment-based communication, a platoon is able to decline platooning requests. Depending on the given situation, a platoon can either decide to drive in a static (no platoon reconfiguration allowed) or in a dynamic platoon. The decision to accept or decline new members is based on decentralized decision making among the current platoon members.

7 Appendix

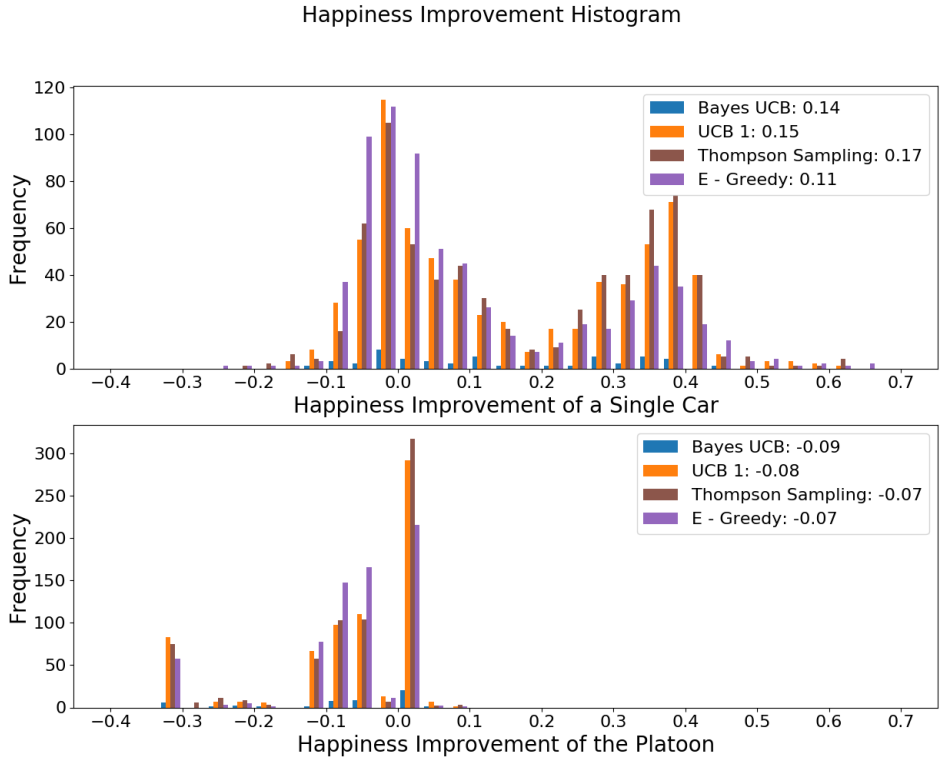

Figure 7.1: Scenario P from table 4.1. Happiness improvement due to platoon changing maneuvers. Plot one shows this improvement for the joining vehicle. Plot two shows the sum of improvement of all members from a platoon, after a new car has joined it. The number in the legend is the average improvement per change.

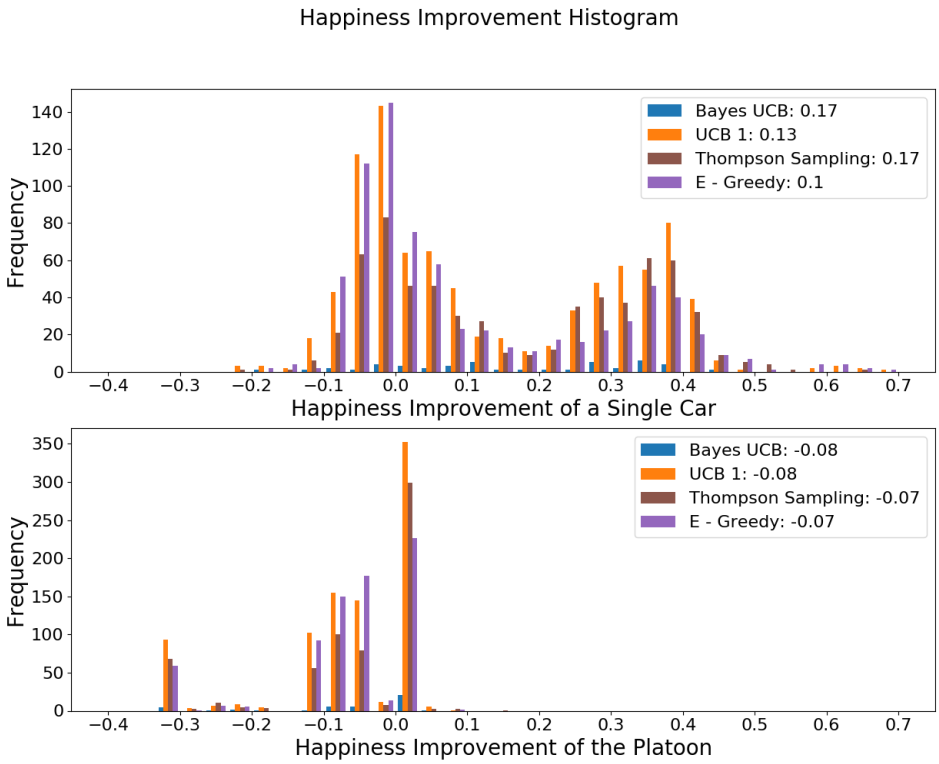

Figure 7.2: Scenario Q from table 4.1. Happiness improvement due to platoon changing maneuvers. Plot one shows this improvement for the joining vehicle. Plot two shows the sum of improvement of all members from a platoon, after a new car has joined it. The number in the legend is the average improvement per change.

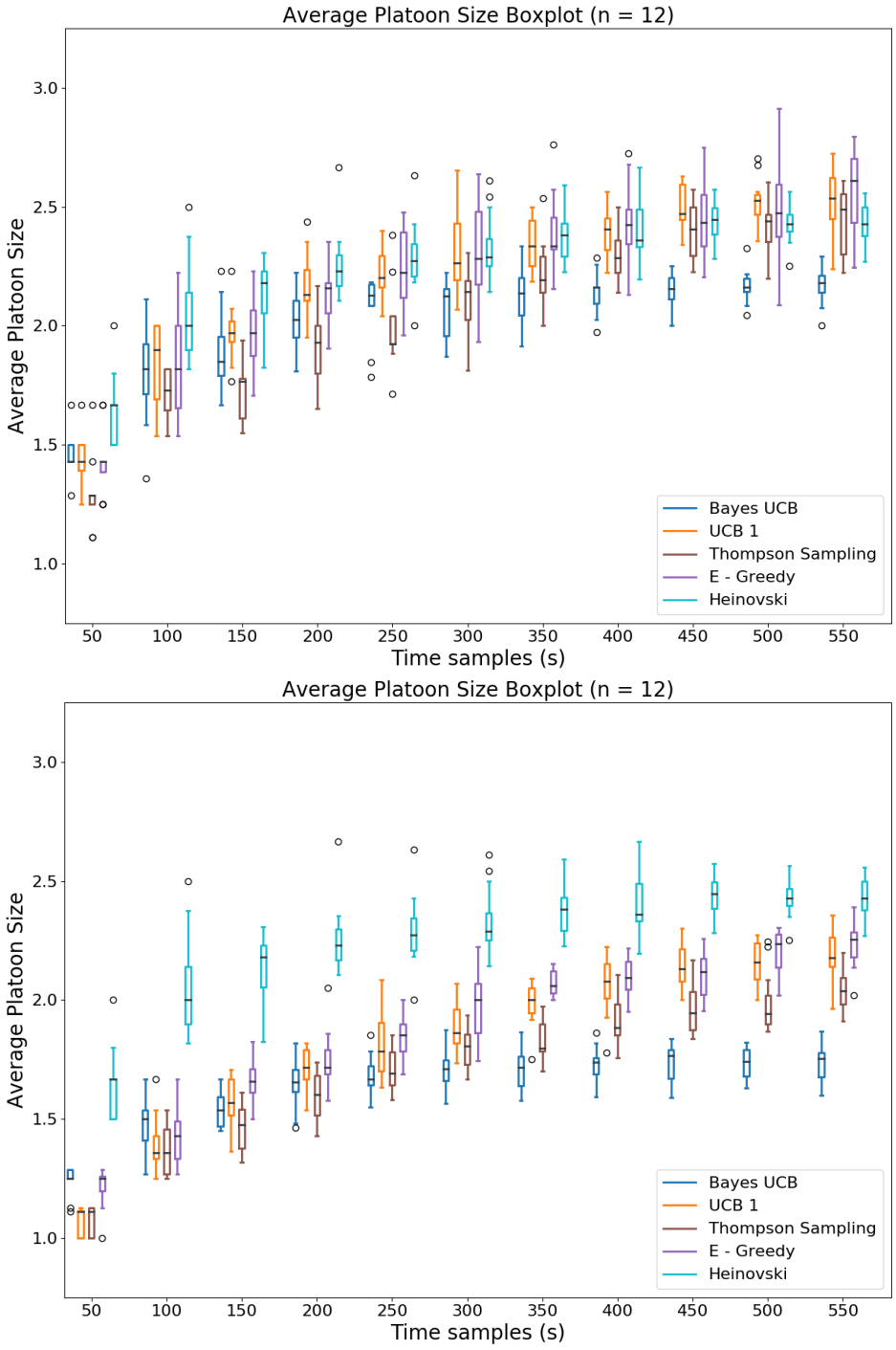

Figure 7.3: Scenario I (upper) and M (below) from table 4.1. Comparison of different decision thresholds w in a *start to end scenario*. Platoon switching threshold $t = 0.04$ is fixed. Speed up is deactivated. First diagram shows scenario with $w = 5$. The scenario below is simulated with $w = 10$.

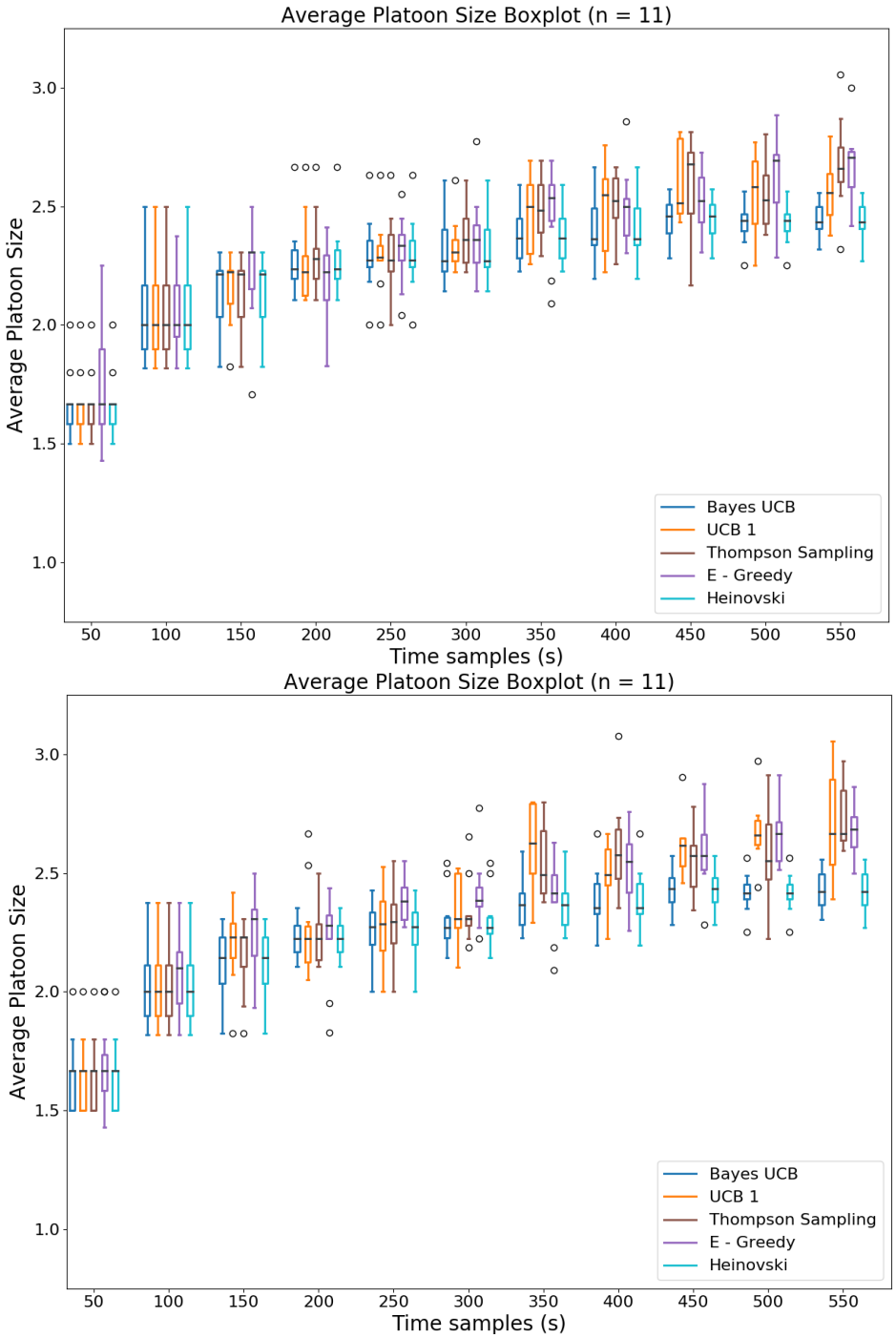

Figure 7.4: Scenario N (upper) and O (below) from table 4.1. Comparison of different platoon change thresholds t in a *start to end scenario*. First diagram shows a scenario with platoon switching threshold $t = 0.01$ and the second one is simulated with $t = 0.02$.

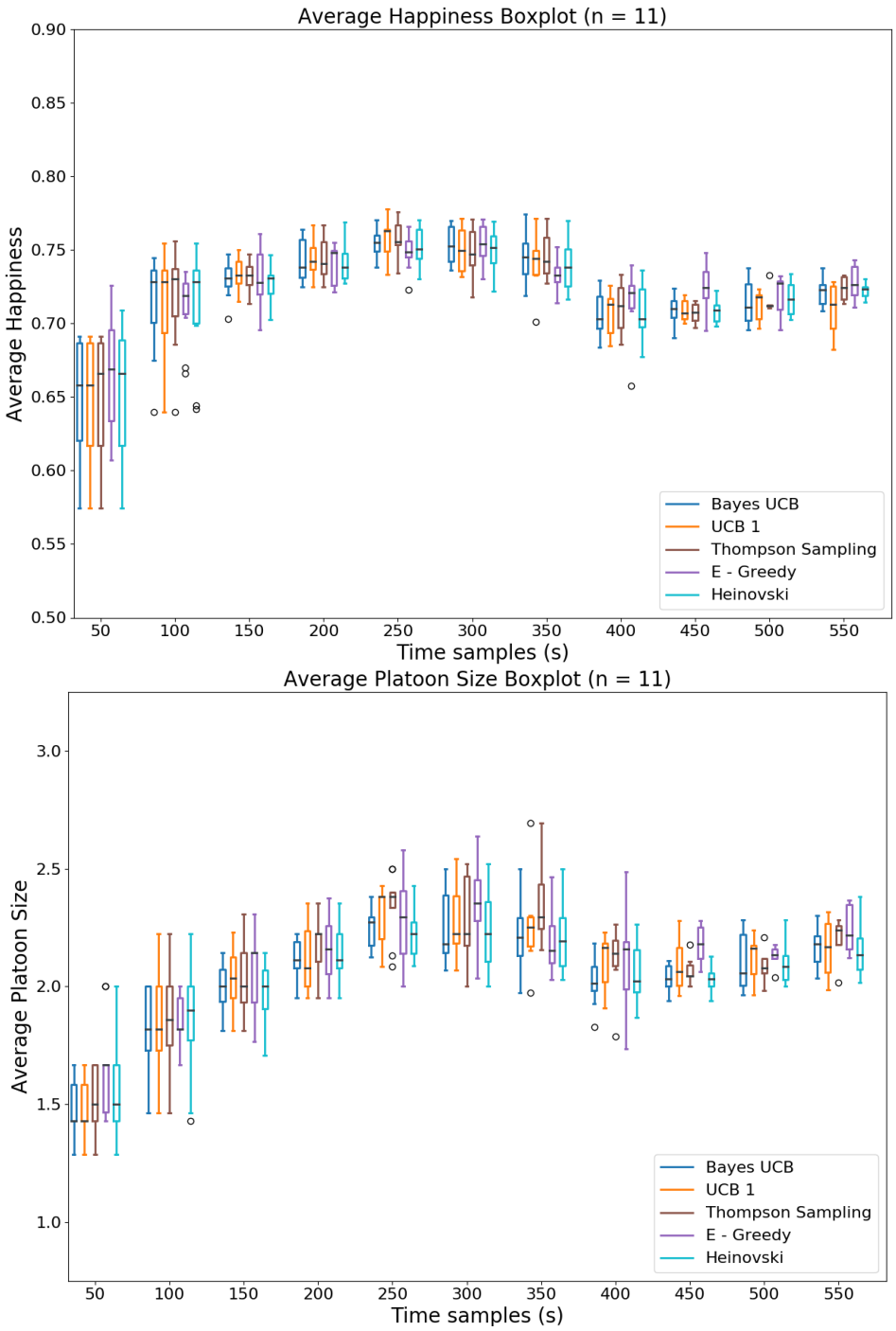

Figure 7.5: Average happiness and average platoon size of a *half-second-scenario*. In this scenario, every half a second, the neighbor calculation algorithms are performing. The frequency of all other scenarios of this work is one second. *start to end scenario*, $w = 5$, $t = 0$, activated speed up.

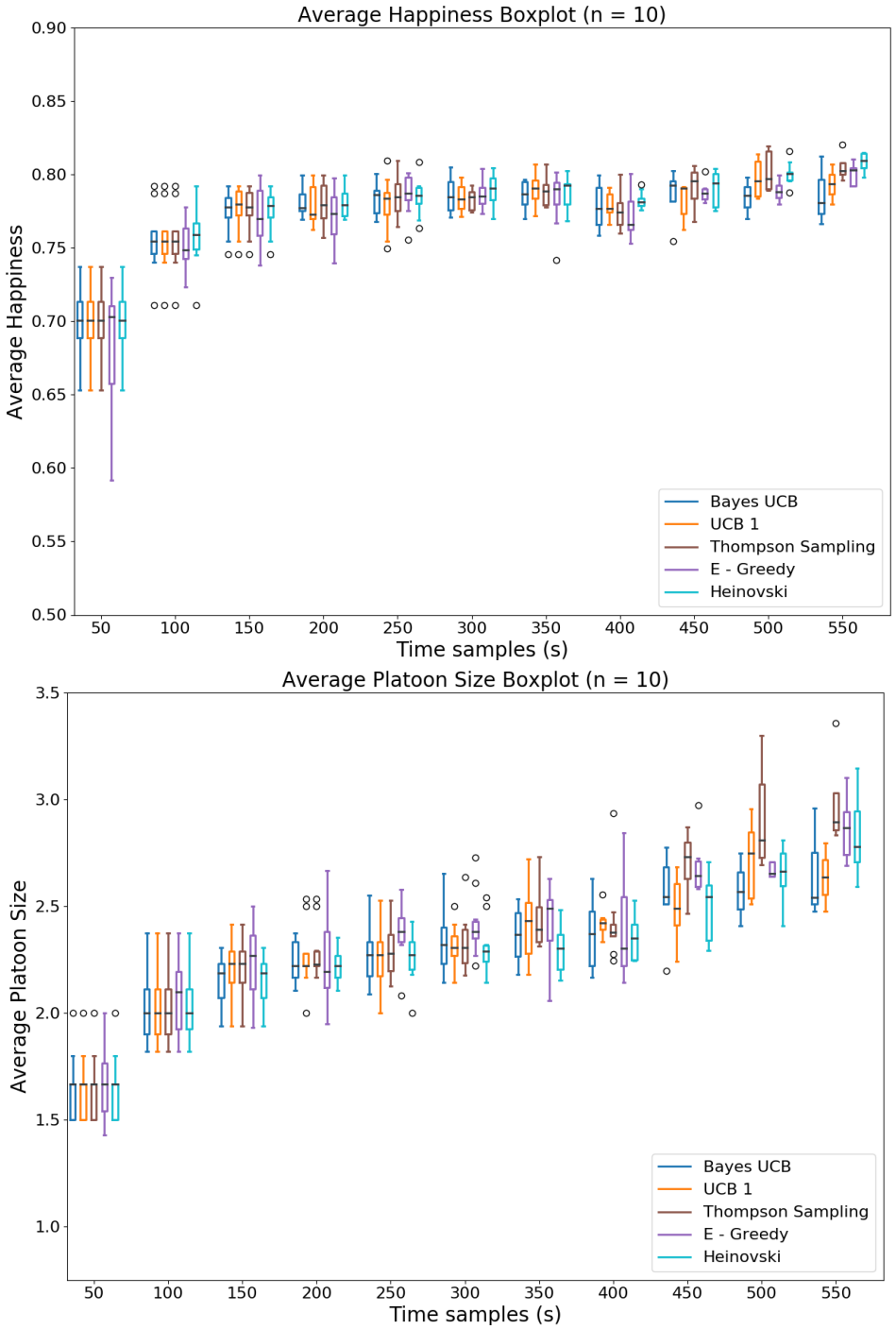

Figure 7.6: Average happiness and average platoon size. *Decision threshold* $w = 1$ is set. The other parameters are identical to scenario H from table 4.1. *Start to end scenario*, $t = 0$, activated *speed up*.

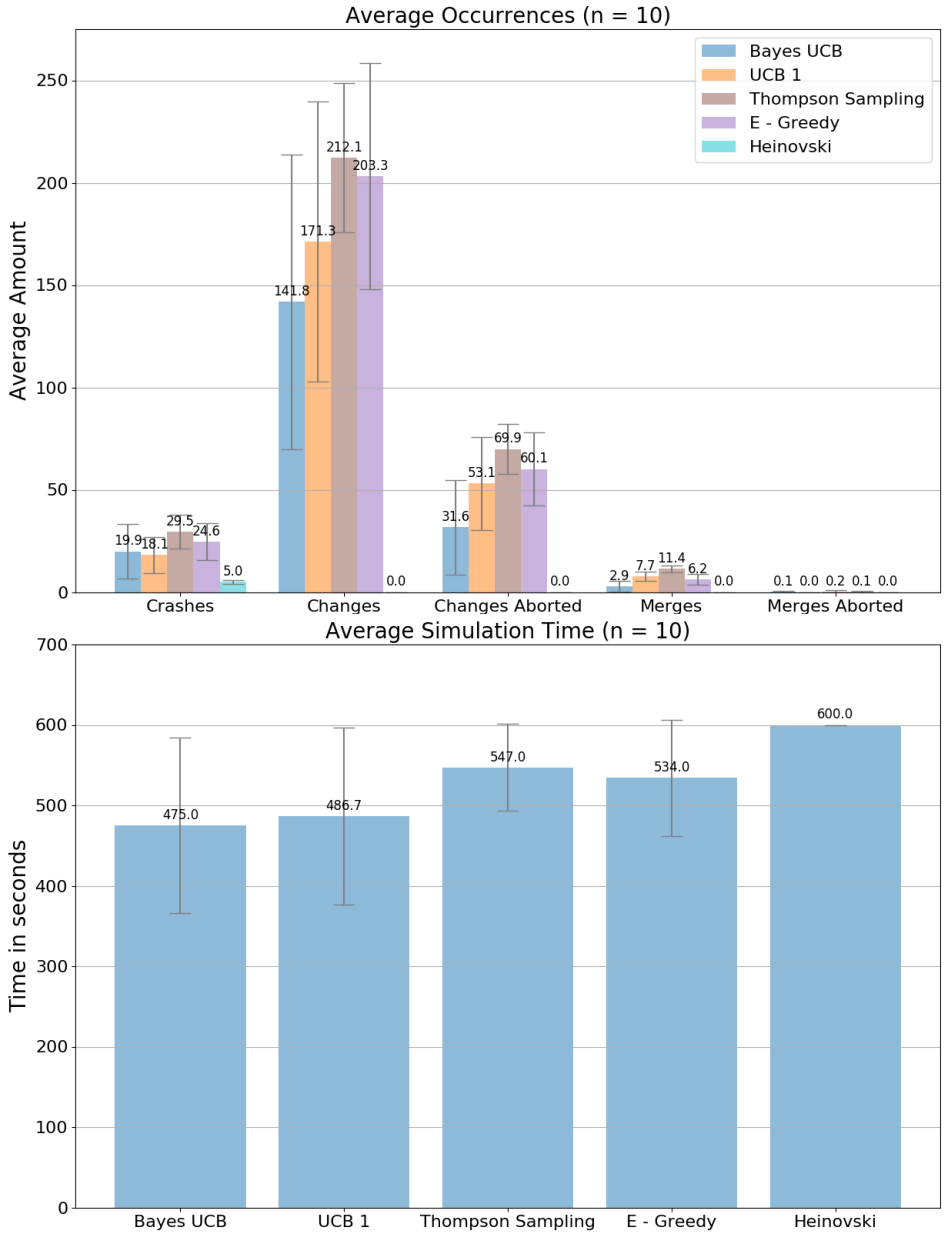

Figure 7.7: Average occurrences and average simulation time. *Decision threshold* $w = 1$ is set. The other parameters are identical to scenario H from table 4.1. *Start to end scenario*, $t = 0$, activated *speed up*.

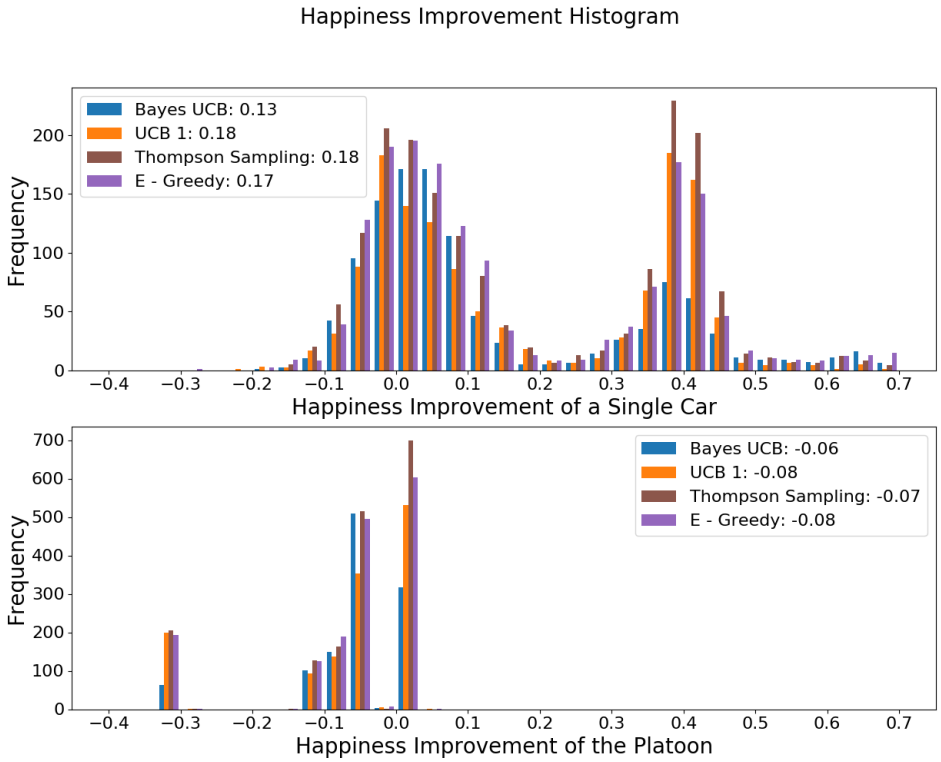

Figure 7.8: Platoon switches histogram. *Decision threshold* $w = 1$ is set. The other parameters are identical to scenario H from table 4.1. Happiness improvement due to platoon changing maneuvers. Plot one shows this improvement for the joining vehicle. Plot two shows the sum of improvement of all members from a platoon, after a new car has joined it. The number in the legend is the average improvement per change.

Bibliography

[1] Carl Bergenhem, Steven Shladover, Erik Coelingh, Christoffer Englund, and Sadayuki Tsugawa. Overview of platooning systems. In *Proceedings of the 19th ITS World Congress*, 2012.

[2] C. Braun. *Computernetze kompakt, Kapitel 3*. Springer Berlin Heidelberg, 2017. ISBN 978-3-662-59896-2. URL `https://books.google.de/books?id=2WqYDgAAQBAJ`.

[3] R. Brause. *Betriebssysteme: Grundlagen und Konzepte*. Springer Vieweg Berlin Heidelberg, 2018. ISBN 9783662541005. URL `https://www.springer.com/de/book/9783662574690`.

[4] Bundesanstalt für Straßenwesen. Rechtsfolgen zunehmender Fahrzeugautomatisierung. URL `https://www.bast.de/BASt_2017/DE/Publikationen/Foko/2013-2012/2012-11.html`.

[5] Bundesministerium für Verkehr und digitale Infrastruktur. Mobilität in Deutschland - MiD. URL `https://www.bmvi.de/SharedDocs/DE/Anlage/G/mid-ergebnisbericht.pdf?__blob=publicationFile`.

[6] M. M. Caballeros Morales, C. S. Hong, and Y. C. Bang. An adapt- able mobility-aware clustering algorithm in vehicular networks. *IEEE, Sep. 2011, pp. 1–6*, 2011.

[7] German Aerospace Center. Simulation of Urban MObility - Documentation. url=https://sumo.dlr.de/docs/, 2001. [Online; accessed 03-Januar-2020].

[8] David Silver. UCL Course on RL, Lecture 9: Exploration and Exploitation. URL `https://www.davidsilver.uk/teaching/`.

[9] Arturo Davila, Eduardo del Pozo, Enric Aramburu, and Alex Freixas. Environmental benefits of vehicle platooning. In *SAE Technical Paper*. The Automotive Research Association of India, 01 2013. doi: 10.4271/2013-26-0142. URL `https://doi.org/10.4271/2013-26-0142`.

[10] Federal Highway Research Institute of Germany. GESCHWINDIGKEITEN AUF BUNDESAUTOBAHNEN IN DEN JAHREN 2010 BIS 2014, 2016. URL `https://www.bast.de/BASt_2017/DE/Publikationen/Fachveroeffentlichungen/Fachveroeffentlichungen-V_node.html`.

[11] Hamburger Hafen und Logistik AG. ALTENWERDER (CTA). URL `https://hhla.de/de/container/altenwerder-cta/so-funktioniert-cta.html`.

[12] Cédric Hartland, Sylvain Gelly, Nicolas Baskiotis, Olivier Teytaud, and Michèle Sebag. Multi-armed bandit, dynamic environments and meta-bandits. 11 2006.

[13] Julian Heinovski. Investigating strategies for building platoons of cars. *Master's thesis at University of Paderborn, GER*, 2018. URL `http://digital.ub.upb.de/hs/content/pageview/2944577`.

[14] Julian Heinovski and Falko Dressler. Platoon formation: Optimized car to platoon assignment strategies and protocols. In *2018 IEEE Vehicular Networking Conference (VNC)*, pages 1–8, Dec 2018. doi: 10.1109/VNC.2018.8628396. URL `https://doi.org/10.1109/VNC.2018.8628396`.

[15] L.H.X. Hobert. A study on platoon formations and reliable communication in vehicle platoons. *Master's thesis at University of Twente, NL*, 2012. URL `https://www.utwente.nl/en/eemcs/dacs/assignments/completed/master/reports/2012_hobert.pdf`.

[16] A. Johnsen, N. Strand, J. Andersson, C. Patten, C. Kraetsch, and J. Takman. Literature review on the acceptance and road safety, ethical, legal, social and economic implications of automated vehicles. *Materialien aus dem Institut für empirische Soziologie an der Friedrich-Alexander-Universität Erlangen-Nürnberg*, 2018. URL `https://www.ifes.fau.de/referenzen/publikationen/ifespubl-mat/`.

[17] Toshiaki Kakinami, Mitsuyoshi Saiki, and Jun Sato. Vehicle cruise control system - . Patent US5230400A, 1990. URL `https://patents.google.com/patent/US5230400A/en#citedBy`.

[18] J. Larson, K. Y. Liang, and K. H. Johansson. Distributed framework for coordinated heavy-duty vehicle platooning. *MIEEE Transactions on Intelligent Transportation Systems, vol. 16, no. 1, pp. 419–429*, 2015.

[19] K. Y. Liang, J. Mårtensson, and K. H. Johansson. Heavy-duty vehicle platoon formation for fuel efficiency. *MIEEE Transactions on Intelligent Transportation Systems vol. 17, no. 4, pp. 1051–1061*, 2016.

[20] Markus Maurer, J. Gerdes, Barbara Lenz, and Hermann Winner. *Autonomous Driving. Technical, Legal and Social Aspects*. 05 2016. doi: 10.1007/978-3-662-48847-8.

[21] Nuremberg City. Echtes Pionierstück: Nürnbergs automatische U-Bahn. URL `https://www.nuernberg.de/internet/digitales_nuernberg/automatische_ubahn_nuernberg.html#4`.

[22] C. Pelekis and J. Ramon. Hoeffding's inequality for sums of dependent random variables. *Mediterranean Journal of Mathematics*, page 243, 2017. ISSN 1660-5454. doi: 10.1007/s00009-017-1043-2. URL `https://doi.org/10.1007/s00009-017-1043-2`.

[23] Prognos AG. Einführung von Automatisierungsfunktionen in der Pkw-Flotte - Auswirkungen auf Bestand und Sicherheit, 2018. URL `https://www.adac.de/rund-ums-fahrzeug/ausstattung-technik-zubehoer/autonomes-fahren/technik-vernetzung/aktuelle-technik/`.

[24] Hamed Raza and Petros Ioannou. Vehicle following control design for automated highway systems. volume 2, pages 904 – 908 vol.2, 06 1997. ISBN 0-7803-3659-3. doi: 10.1109/VETEC.1997.600460.

[25] Tom Robinson, Eric Chan, and Erik Coelingh. An introduction to the sartre platooning programme. In *Proceedings of the 2010 ITS World Congress*, 2010.

[26] Daniel J. Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on thompson sampling. *Found. Trends Mach. Learn.*, 11(1): 1–96, July 2018. ISSN 1935-8237. doi: 10.1561/2200000070. URL `https://doi.org/10.1561/2200000070`.

[27] M. Segata, S. Joerer, B. Bloessl, C. Sommer, F. Dressler, and R. L. Cigno. Plexe: A platooning extension for Veins. In *IEEE Vehicular Networking Conference (VNC)*, pages 53–60, Dec 2014. doi: 10.1109/VNC.2014.7013309. URL `https://doi.org/10.1109/VNC.2014.7013309`.

[28] Steven Shladover, Xiao-Yun Lu, Shiyan Yang, Hani Ramezani, John Spring, Christopher Nowakowski, and David Nelson. Cooperative Adaptive Cruise Control (CACC) For Partially Automated Truck Platooning:Final Report. Institute of Transportation Studies, Research Reports, Working Papers, Proceedings qt260060w4, Institute of Transportation Studies, UC Berkeley, August 2018. URL `https://ideas.repec.org/p/cdl/itsrrp/qt260060w4.html`.

[29] Aleksandrs Slivkins. Introduction to Multi-Armed Bandits. *arXiv e-prints*, art. arXiv:1904.07272, April 2019.

[30] Statistisches Bundesamt (Destatis). *Statistisches Jahrbuch Deutschland und Internationales 2019*. 10 2019. doi: Artikelnummer1010110-19700-4.

[31] S. Tsugawa, S. Kato, and K. Aoki. An automated truck platoon for energy saving. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4109–4114, 2011.

[32] Chaojie Wang, Siyuan Gong, Anye Zhou, Tao Li, and Srinivas Peeta. Cooperative adaptive cruise control for connected autonomous vehicles by factoring communication-related constraints. *Transportation Research Procedia*, 38:242 – 262, 2019. ISSN 2352-1465. doi: https://doi.org/10.1016/j.trpro.2019.05.014. URL `http://www.sciencedirect.com/science/article/pii/S2352146519300237`. Journal of Transportation and Traffic Theory.

[33] T. L. Willke, P. Tientrakool, and N. F. Maxemchuk. A survey of inter-vehicle communication protocols and their applications. *IEEE Communications Surveys Tutorials*, 11(2):3–20, Second 2009. ISSN 2373-745X. doi: 10.1109/SURV.2009.090202.