INTELLIGENT TRANSPORTATION SYSTEMS: ALGORITHMS AND FORMAL VERIFICATION

Ph.D. Thesis

By

AADITYA PRAKASH CHOUHAN

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

NOVEMBER 2020

INTELLIGENT TRANSPORTATION SYSTEMS: ALGORITHMS AND FORMAL VERIFICATION

A THESIS

submitted to the

INDIAN INSTITUTE OF TECHNOLOGY INDORE

in partial fulfillment of the requirements for
the award of the degree of

DOCTOR OF PHILOSOPHY

by

AADITYA PRAKASH CHOUHAN

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

NOVEMBER 2020

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **Intelligent Transportation Systems: Algorithms and Formal Verification** in the partial fulfillment of the requirements for the award of the degree of **Doctor of Philosophy** and submitted in the **Discipline of Computer Science and Engineering, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the time period from December 2015 to November 2020 under the supervision of Dr. Gourinath Banda, Associate Professor, Indian Institute of Technology Indore, Indore, India.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

19/4/2021

Signature of the Student with Date

(Aaditya Prakash Chouhan)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Signature of Thesis Supervisor with Date

(Dr. Gourinath Banda) 19/04/2021

Aaditya Prakash Chouhan has successfully given his Ph.D. Oral Examination held on **April 19, 2021**

Signature of Thesis Supervisor with Date

(Dr. Gourinath Banda) 19/04/2021

ACKNOWLEDGEMENTS

I wish to seize this opportunity to acknowledge all who have assisted, one way or another, to the accomplishment of this dissertation, which marks another important milestone in my life.

First and foremost, the sincerest gratitude ascribes to my supervisor Dr. Gourinath Banda for his immense support and unrelenting source of motivation throughout my PhD tenure. His benevolent supervision, constructive criticism, and invaluable guidance from the moment I started working at IIT Indore as a doctoral student. The consistent and unending directions steered me in the right path and transformed me into a researcher who can work independently. I could not have imagined having a better, affable supervisor and mentor for my doctoral tenure.

Besides, I extend my gratitude to my research progress committee members: Dr. Neminath Hubballi and Dr. Vimal Bhatia, for their keen observation, valuable comments, insightful suggestions, encouragement, and validation of this research. I would like to express my appreciation to Dr. Somnath Dey, Head of Discipline for all his extended suggestions and support.

I express sincere gratitude to Prof. Neelesh Kumar Jain (Director, IIT Indore), who has been highly encouraging during the entire course of my doctoral work. A special gratitude goes to Indian Institute of Technology Indore for providing me an opportunity to pursue PhD under state-of-the-art research environment and also Ministry of Human Resource Development (MHRD), Government of India for granting financial aid to carry out this doctoral study successfully.

I am also grateful to cheerful doctoral fellows; Aditya Shastri, Rohit Agarwal, Navneet Pratap Singh, Dr. Ram Prakash, Dr. Chandan Gautam, Dr. Piyush Joshi, Dr. Nikhil Tripathi as well as the undergraduate students for their feedback, encouragement, cooperation, and of course friendship. Without their precious support, it would not be possible for me to reach this level. Also, I would wish to thank the all staffs of Computer Science and Engineering Discipline.

I must express my profound gratitude to my family: my parents, for providing me with unfailing support and continuous encouragement throughout my years of study through the process of doctoral research and writing this thesis, and my better half, Ayushi, for her love and support. Their belief in me and my capabilities is the prime motivation that has driven me to this stage where I can realize my dreams. This accomplishment would not have been possible without them.

Aaditya Prakash Chouhan
Indian Institute of Technology Indore
November 2020

Dedicated to My Parents

Abstract

The adoption of Connected and Automated Vehicles (CAV) on public roads along with offering freedom to humans, offers immense possibilities for devising intelligent strategies for traffic management that can optimally utilize their precise control over movement, ability to communicate and cooperate, consistency, etc. By utilizing these qualities of CAVs enhancements in traffic management can be achieved with respect to efficiency, safety, and economy. This dissertation documents our efforts to utilize the capabilities of CAVs for their efficient management while performing collective (or individual) lane changing operations and at intersections.

This dissertation presents contributions to the field of Intelligent Transportation Systems (ITS) by proposing novel algorithms for multiple ITS scenarios. First, we introduce a new ITS procedure called Lane Sorting, which refers to the rearrangement of vehicles such that every vehicle gets into its desired lane. We then propose the Cooperative Lane Sorting (CLS) algorithm, which is a cooperative algorithm generalized in terms of the number of lanes and traffic density. CLS process vehicles in independent batches called frames, and processing each frame involves solving a non-linear optimization problem reducible to a mixed-integer linear programming problem (MILP). After obtaining a solution to the MILP, vehicles adjust their position and perform the lane change operation. Being a generalized algorithm, CLS can be used for performing lane change operation of any number of vehicles wanting to do so in a given group of vehicles.

Next, we propose the Heuristic Autonomous Intersection Management (HAIM) algorithm for intersection management of autonomous vehicles. HAIM makes use of four levels of heuristics to schedule vehicles such that no two vehicles are present at the same space at the same time and the delay caused in this scheduling is minimum. The HAIM algorithm is proposed for a scenario that is equipped with a central controller along with roadside infrastructure for wireless communication. The central controller called Intersection Manager (IM) communicates with all incoming vehicles to gather information regarding their route, vehicle parameters, and motion parameters. The IM, which implements the HAIM algorithm, will then return each vehicle the velocity with which it has to travel. Vehicles, as soon as they enter the scenario, are required to transit to the given velocity and then maintain it throughout the rest of the scenario. The HAIM algorithm works on traffic that already contains vehicles

on lanes corresponding to their destination direction at the intersection. Added with the fixed trajectory of vehicles at the intersection, the problem of intersection management reduces to the scheduling of vehicles at some fixed number of conflict points thus reducing the complexity of the overall problem. A comparative study of the HAIM algorithm with two other autonomous intersection management algorithms and also with traffic light control is later performed. Results obtained show that of all considered intersection management schemes, the HAIM algorithm introduces the minimum delay in vehicle trips caused by scheduling.

The next major contribution of this dissertation is to the field of formal verification of ITS algorithms. We perform verification of the design and operation of the HAIM algorithm and through this, we document an innovative approach to the verification of ITS algorithms that can be performed using available tools and techniques. For verification of HAIM, formal modeling of the algorithm, traffic injection, vehicle behavior, and collision detection is performed and the models so obtained are scrutinized using a combination of techniques. Along with verifying the safety properties, sanity of the modeling, and verification of the expected behaviors of each of the four layers of HAIM are also performed. Techniques such as statistical model checking, simulation testing, internal verification, and artificial error injection are used for this purpose. The approach used to verify the safety property of HAIM and sanity of modeling is unique in terms of the combination of techniques used.

Email: phd1501201006@iiti.ac.in

List of Publications

A. Published

[1] **Aaditya Prakash Chouhan** and Gourinath Banda, “Autonomous Intersection Management: A Heuristic Approach”, *IEEE-Access*, Vol. 6, pp. 53287–53295, 2018, DOI = "https://doi.org/10.1109/ACCESS.2018.2871337", (SCI, IF:4.098)

[2] **Aaditya Prakash Chouhan**, Gourinath Banda and Kanishkar Jothibasu, “A Cooperative Algorithm for Lane Sorting of Autonomous Vehicles”, *IEEE-Access*, Vol. 8, pp. 88759 - 88768, 2020, DOI = "https://doi.org/10.1109/ACCESS.2020.2993200", (SCI, IF: 4.098)

[3] **Aaditya Prakash Chouhan**, Gourinath Banda, “Formal Verification of Heuristic Autonomous Intersection Management Using Statistical Model Checking”, *Sensors*, Vol. 20, 2020, DOI = "https://doi.org/10.3390/s20164506", (SCI, IF: 3.031)

Contents

List of Figures

List of Tables

List of Abbreviations & Acronyms

ABS	Anti-locking Braking System
ADAS	Advanced Driving Assistance Systems
ADS	Automated Driving Systems
AIM	Autonomous Intersection Management
AHS	Automated Highway System
AV	Autonomous Vehicle
CAV	Connected and Autonomous Vehicle
CLC	Cooperative Lane Change
CLS	Cooperative Lane Sorting
CTL	Computational Tree Logic
DARPA	Defense Advanced Research Projects Agency
DSRC	Dedicated Short Range Communication
FCFS	First Come First Serve
GSM	Global System for Mobile Communication
HAIM	Heuristic Autonomous Intersection Management
IEEE	Institute of Electrical and Electronics Engineers
IM	Intersection Manager
INR	Indian Rupee
ITS	Intelligent Transportation Systems
LSA	Lane Sorting Area
MILP	Mixed Integer Linear Programming
PATH	Partners for Advanced Transit and Highways
RSU	Road Side Unit
SAE	Society of Automotive Engineers

SC	Scenario Controller
SDC	Self Driving Car
SMC	Statistical Model Checking
STA	Stochastic Timed Automata
TA	Timed Automata
USDoT	United States Department of Transportation
V2I	Vehicle to Infrastructure
V2V	Vehicle to Vehicle
VSSA	Voluntary Safety Self-Assessment
VTS	Velocity Transition Section
V&V	Verification and Validation
WAVE	Wireless Access in Vehicular Environments

Chapter 1

Introduction

Automobiles have come a long way since their invention. The history of automobiles starts in 1769 when the first steam-powered automobile capable of human transportation was developed by Nicolas-Joseph Cugnot. However, the year 1866 is regarded as the year of birth of the modern automobile as it marks the year Karl Benz developed a gasoline-powered four-stroke single-cylinder automobile [1]. This is also considered to be the first production vehicle as several similar copies were made by Benz. The early history of automobile development is associated with the exploration of different means of propulsion such as steam-powered, four-stroke petrol internal combustion, electric battery, etc. While the later history of automobile development is defined by the trends in exterior styling, size, and utility preferences. The mass production of automobiles had begun in the early 1900s in France and the United States. Early 20^{th} century saw the rise of a large number of automobile companies. Various famous automobile manufacturers of the present day such as Audi, BMW, Volkswagen, Porsche, Bentley, Jaguar, etc. were established in the first half of the 20^{th} century.

With the growing use of automobiles by common people, manufacturers kept on evolving automobiles in terms of safety, efficiency, design, features, and environmental considerations. For instance, the three-point seat belt that is mostly used today was invented by Nils Bohlin in 1959 while working at Volvo. Volvo later made the patent of the same open in the interest of safety and made it available to other car manufacturers for free. Aerodynamic design, Anti-lock Braking System (ABS), Electronic Traction Control, Automatic Transmission, etc. are some more examples of additions made to the automobiles as they evolved.

Most of the vehicles that are now manufactured contain most of these features. These features are now so common that any vehicle that has some or all of these features can be called a *conventional vehicle*. These *conventional vehicles* are used throughout the world and have become the most important mode of transportation. They have made humans more mobile, increased their productivity, created new opportunities, increased their social and economic interactions, etc. As a result, cars have become an integral part of the lifestyle of a large portion of the world's population. Automobiles have also become a source of income for those who are in a transportation service business such as logistics, taxi operators, drivers; be it bus or truck or cargo vehicles. This tells us that automobiles now play an indispensable role in the social and economic life of people throughout the world.

1.1 The Call for Automated Traffic

A safe and efficient operation of automobiles relies on the availability of various infrastructural resources that contain and manage them. The infrastructural requirement for the operation of automobiles in the most basic form is a road on which they are driven. When we have a network of roads, infrastructures such as intersection, road merge/diverge, flyovers, etc. come into the picture. Generally, these infrastructures have a limit on the traffic (a group of automobiles that are using a common infrastructure) density they can handle at any given time. When the traffic grows larger than the capacity of the infrastructure, it gives rise to Congestion. In such a scenario, the infrastructure does not have the capacity to handle the incoming traffic in an efficient way, and the resulting performance of the traffic control scenario will be sub-optimal. As a result, vehicles move at slower speeds, take longer trip times and there is an increased queuing of vehicles.

Traffic congestion has become a serious problem throughout the world. There are a number of negative effects of traffic congestion which includes:

- It increases the trip time which results in wasted human hours and hence less productivity.

- It results in decreased economy of the vehicle. As a result, increasing the cost of operation of the vehicle by burning more fuel, maintenance cost, etc.

- Wasted fuel also results in more emissions from the vehicle, thus increasing the environmental damage.

- It may result in a feeling of stress and frustration in the driver that may cause him/her to behave irrationally thus increasing the chances of road rage and collisions.

A report [2] says that in the United States alone, drivers have lost more than $88 Billion in the year 2019 due to congestion. A similar study in India shows that traffic congestion has resulted in a total loss of INR 144 thousand crores (1.44 trillion) in just four cities of Mumbai, Delhi, Bangalore, and Kolkata in the year 2017 [3]. Along with economic loss, congestion also hits the environment very hard. Another study says that an average passenger vehicle in the United States emits 4.6 metric tons of carbon dioxide in a year [4]. Urban populations living in developed and developing nations are most affected by these adverse effects of congestion.

Often times, to deal with congestion, the infrastructure is extended or rebuilt to accommodate the traffic requirements. Measures taken by the city or the highway authority include developing alternate routes if there is heavy traffic on one route, building flyovers to keep different traffic streams uninterrupted, widening of roads to add one or more lanes, etc. Though these measures are effective, they can't be used everywhere because of the following reasons:

- Infrastructural development requires large capital investment. In developing countries where the government may be under a tight budget or is having other public services at a higher priority, it shall not be willing to invest such huge amounts.

- Even if the budget is not a problem, there may be a situation where there are space restrictions. Such a problem occurs when the congestion hit area is in a densely occupied urban area where there is no space to widen the road or to build alternate routes.

- Most of the time, building infrastructure is a reactive strategy i.e. it is employed as a result of urgency. Such solutions are not always permanent and the traffic scenario can again become congestion hit due to the increase in population.

This tells us that dealing with congestion needs a smarter approach than building infrastructures. Another approach is to manage the present infrastructure using smart traffic management techniques. The review article given in [5] presents a review of such techniques. These

techniques can alleviate the situation to some extent, however, the unexpected rise in the population will eventually result in persistent congestion.

Along with inconvenience and losses caused by congestion, there is another major factor calling for the need of improving the current traffic situation and that is the injuries and deaths caused by traffic accidents. A World Health Organization report [6] says that there were 1.4 million deaths globally in traffic accidents in the year 2016 alone and globally, on an average, there is one fatality in a traffic accident every 20 seconds!

It will not be an exaggeration if we say that the current transportation system is associated with drawbacks such as: high fatality rate, congestion, environmental degradation, economic loss, etc. There have been consistent efforts to cope up with these issues however, these issues are still present and are growing in the intensity.

All these drawbacks and problems faced by the current traffic management even after consistent attempts to curb them, suggest that there is a need for a concrete solution. Also, for this solution to have long-lasting applicability, it should target the root cause of all these problems. Obtaining such a solution may involve some out of the box approach or an approach that hasn't been tried before. One such approach involves transforming the current composition of traffic by replacing the conventional vehicles with vehicles that are technologically advanced and can have autonomous control over their movement in at least some of the situations.

The objective of this transformation is to reduce dependence on humans for performing dynamic driving tasks. It may sound strange but humans are indeed the most unpredictable part of the current transportation systems as they are prone to getting fatigued, drunk driving, impulsive behavior, slower reflex action, divided attention, etc. Humans are accountable for about 94 to 96% of road accidents [7]. Even if a human driver remains composed, attentive, is not in rush, follows all traffic rules, is not fatigues or drunk, etc., he/she is only capable of making local decisions i.e. the decisions which are in his/her best interest and not in the interest of the overall network.

In other words, the scheme for transforming the current transportation system is to transform the vehicles, which are the smallest entity in it. Transforming the traffic to mainly consisting of such advanced vehicles known as Autonomous Vehicles (AV) or Self-Driving Vehicles (SDC) will give a foundation to a much efficient and effective traffic management

algorithms. We will next discuss in detail what autonomous vehicles are and why they are deemed as the next revolution in the transportation systems domain.

1.2 Autonomous Vehicles

Autonomous Vehicles have autonomous control over their movement in at least some of the driving modes and driving conditions. They are the next class of evolution after the previously defined *conventional vehicles*. AVs have gained a lot of popularity in recent times as they are expected to transform the traffic we see today into very well organized streams of cooperative, coordinated, and obedient vehicles. The AV technology opens up a large number of possibilities, for instance, passengers of an AV could utilize the time of commute as per their will instead of driving; AVs could enable elderly, children and disabled people to commute independently; they could park themselves! and many more. All these advantages are indeed crucial, but they don't include the most significant advantage of autonomous vehicles. The true potential of AVs is unleashed when they are enabled by communication capability. This is because when AVs are enabled with communication capability, then called Connected Autonomous Vehicles (CAV), can collectively behave as a team in any traffic scenario to behave in the most efficient way possible. The behavior of CAVs in a traffic scenario can be dictated by a Scenario Controller (SC) which performs all the necessary mathematical processing to come up with the optimum driving strategy for each vehicle or vehicles can even come up with a strategy among themselves without any intervention of any controller. This level of coordination and control is not practically possible with manually driven vehicles. Also, every CAV can be individually controlled by the SC enabling micro-control over traffic in any scenario.

It is not necessary for an AV to have complete autonomy i.e. it need not be a vehicle without steering, throttle, and brakes to be called an AV. An AV can have different levels of autonomy depending on the functions performed by the autonomous driving system of the vehicle. The Society of Automotive Engineers (SAE) defines six levels of autonomy (level 0 to 6) for AVs. We will have a detailed discussion on these levels in Section 2.1 of this dissertation.

Similarly it is not required to have AV with complete autonomy to exploit their capabilities for effective traffic management. We can design algorithms for AVs with intermediate

levels of autonomy as well and leverage their capabilities. This fact has been one of the prime motivations for the works presented in this thesis as these works may find applications in the very near future due to the availability of vehicles on which these algorithms can be applied (level-2 or more autonomous vehicles).

1.3 Intelligent Transportation System

Intelligent Transportation Systems or ITS, refers to traffic management systems that for traffic management, exploit the advancements in technologies such as sensing, wireless communication, computation, vehicle automation, etc. ITS applications aim to provide traffic management services by leveraging real-time information and coordinated vehicle movement in dedicated algorithms for more safe and efficient management of transport networks. Any ITS can be thought of as comprising of four players, namely: i) Sensors, ii) Decision making agents, iii) Vehicle agents and iv) Communication infrastructure. The decision-making agent can be a central controller which is common for all the vehicles in a particular scenario or it can well be a controller station common to vehicles in a region or city, it can also be one of the vehicles in the scenario that has been elected by some algorithm to make local decisions. Vehicle agents in an ITS scenario are required to have functionalities that enable them to communicate with the decision-making agent and can also have autonomous control features.

In an ITS, wireless communication is usually performed between Vehicle and Infrastructure known as Vehicle to Infrastructure (V2I) communication and between different vehicles known as Vehicle to Vehicle (V2V) communication. Various technologies have been proposed to perform V2V and V2I wireless communications. For short-range communication of up to 350 meters, IEEE 802.11p standard can be used which is a standard to add Wireless Access in Vehicular Environments (WAVE). This standard is backed by the United States Department of Transportation (USDoT) and also is the basis of the Dynamic Short Range Communication project (DSRC) run by it. For long-range communication, the available technologies are IEEE 802.16 (WiMAX) protocol, Global System for Mobile Communication (GSM), 3G, 4G, etc.

We have talked about the extent of possibilities that are opened up by CAVs in terms of

traffic control strategies. This is the reason, ITS has become a hot topic among researchers from industries and academies in the last decade. There have been various government initiated programs as well to bring together researchers from across the globe and showcase their technology. These attempts have been directed towards both autonomous vehicle developments as well as towards their application in ITS scenarios. For instance, the Automated Highway System (AHS) was developed at the University of California in association with California state department of transportation and the Federal Highway Administration under the PATH (Partners for Advanced Transportation Technology) program [8]. Similarly, in the early 2000s, the U.S. defense organization, DARPA, conducted multiple challenges [9] to accelerate the autonomous technology for military requirements. This turned out to be a path-breaking program as it gave momentum to the research on autonomous driving technology. In addition to this, lots of large automobile companies and various startups have heavily invested in CAVs and are working towards their autonomous vehicles and/or vehicles with Advanced Driving Assistance Systems (ADAS).

The number of CAV with respect to the total number of vehicles, also known as the penetration rate (PR) is a measure of the density of the CAVs in a city or a region. The extent of benefits experienced in the traffic conditions by adopting CAVs depends on the penetration rate. As the number of CAVs increases relative to the number of conventional vehicles, or in other words, as the penetration rate of CAVs increases, most of the problems occurring in the current traffic situation will be diluted and the efficiency of transportation systems would improve. As the advantages of using CAVs are huge, it is very important to develop traffic management algorithms that take advantage of the connectivity and autonomous control. Although many efforts have been directed in this direction, still the development of safe and efficient traffic management algorithms for CAVs has not matured. As a result, devising efficient algorithms for various traffic scenarios remains an open research problem. Few examples of such traffic scenarios are Intersections, Lane merges, Highway ramps, Toll gates, Roundabouts, etc. Among these and various other scenarios whose traffic control strategies are crucial, in this dissertation, we are particularly interested in the autonomous intersection control and *Lane Sorting* of CAVs.

In addition to proposing ITS algorithms, we are also interested in procedures to generate correctness proof for the design and behavior of such algorithms against the required properties.

Because of the safety critical nature of these algorithms, it is crucial to obtain such proof. We make use of formal verification for this purpose, which is a mathematically backed technique. We will next discuss about the need of correctness verification (which influences the overall safety of the product/technology) and some industry practices for the same.

1.4 Formal Verification

The role that technology plays in human life has grown over the years. Out of all the technologies that surround humans, some of them can have direct consequences on the safety of the human user, for instance, elevators, driver assistance features, auto-pilot systems in an airplane, etc. For such technologies, it is required to have strict Verification and Validation (V&V) procedures to ensure safety of users post deployment. Along with the impact on the safety of the users, V&V procedures are also critical in preventing economical losses. In practice, it is well known that the earlier the errors are found in the product development cycle, the lower is the cost of correcting such errors.

In Validation and Verification (V&V) procedures, validation is targeted towards guaranteeing that the correct product is developed, whereas, the verification techniques are targeted towards guaranteeing that the product is developed in the right way. Common V&V techniques are formal verification, testing, simulation, etc. Simulation, as applied for the purpose of V&V is a technique that performs executions of the model of the system under development for some finite number of test cases. Testing, on the other hand, is applied after a system is past its development stage. Even though simulation and testing are indispensable in V&V procedures, they have their limitations. Simulations can only have coverage of the state-space corresponding to the test-cases considered and testing can only detect the presence of errors and not the absence of them [10]. Formal verification is a mathematical approach for guaranteeing that the system model satisfies the specified requirements. Broadly, such system requirements (also called as system properties) belong to two classes: (a) safety properties and (b) liveness properties. Safety properties are of the form "Something bad will never happen", whereas liveness properties are of the form "Something good will eventually happen". These two types of properties can be combined to specify other requirements [11]. Formal verification techniques makes use of state exploration procedures

and has the capability to comment decisively on the correctness of the given system with respect to specifications.

Formal verification is a well-known technique and has been in use in hardware as well as software verification for a long time now. It is used in most of the hardware companies as errors in the hardware production cycle has much significant commercial influence as compared to that of software. For instance, the car manufacturing company Toyota had to recall 2.4 million cars to fix the issue of loss of power in rare cases due to a fault in the inverter system in the cars manufactured in the year 2008 to 2014 [12]. Similarly, Intel had to recall all the defective processors because of a hardware bug infamously known as *Pentium FDIV bug* [13].

Owing to the capabilities of formal verification techniques, its usefulness in the development of autonomous vehicles and associated algorithms is irreplaceable. It is going to play a crucial role at each stage in the development cycle of such systems. In fact, various autonomous vehicles standards and guidelines recommend formal verification based correctness generation. For instance, ISO26262, which is an international standard for functional safety of electrical and electronic systems of autonomous vehicles uses a system of steps to regulate the development at the system, hardware, and software level. It defines the automotive safety life cycle (management, development, production, operation, service, decommissioning), provides a risk-based analysis for determining risk classes, and provides the requirement of validation measures to ensure the reduction of risk factors to acceptable levels.

The United States Department of Transportation (U.S. DoT) has acknowledged the concerns of the general public regarding the safety of autonomous vehicles and issued guidelines for AV developers, manufacturers, and researchers to help address the issue and make this technology safe for wide acceptance by the public. They published their latest guidelines for developing a safe autonomous vehicle technology named *Preparing for the future of transportation 3.0* [14]. The guideline is aimed towards mitigating risks introduced by the automation of vehicles and thus be able to exploit the full potential of the technology. The guidelines issued in this latest document extends the previous guideline named *A vision for safety 2.0* [15] that emphasized on the safety aspect of Automated Driving System (ADS) and kept testing, verification/validation central to the U.S. DoT's approach. ADS manufacturers

are also encouraged to make their safety assessment report that includes the description of the approach used for verification/validation named Voluntary Safety Self-Assessment (VSSA) public in order to increase transparency and confidence in their technology.

Because of the safety critical nature of algorithms that control vehicles in ITS scenarios, it is crucial to obtain their correctness proofs to ensure safety of lives. Generating such proofs, besides improving in algorithm's correctness confidence will also be useful in terms of certification and gaining trust of general public. This motivates us to propose a verification technique for the proposed intersection management algorithm. The key property of the algorithm which we consider is *collision freedom*.

1.5 Problems Considered and Motivations

The works presented in this thesis make contributions to the ITS domain and formal verification domain. We first propose one algorithm each for: (i) cooperative and coordinated lane changes of a group of AVs and (ii) autonomous intersection management. Thereafter, we present a systematic approach for verifying the collision freedom property of the proposed autonomous intersection management algorithm. The research carried out has been primarily motivated by the following observations.

1.5.1 Lane Sorting: Cooperative rearrangement of vehicles in lanes

Lane change is one of the most common maneuvers involved in vehicular driving. It is performed while overtaking, at highway entry or exits, merges, etc. In the literature there are several works dedicated to planning safe and efficient lane change maneuvers for individual vehicles [16]. Also, there exist works that make use of lane changing operations to organize vehicles into arrangements that result in an efficient flow of the traffic. Platooning is one such technique [17]. Despite the role that lane change operations play in traffic management, we have observed that there is no dedicated work in the literature that aims to plan cooperative lane change operations for a group of vehicles. Vehicles are assigned their most favorable lane by lane assignment techniques [18] but to the best of our knowledge, planning how vehicles can physically coordinate their lane change operations such that all vehicles reach their destination lanes within minimum distance and without any conflict is one area that hasn't

been approached as a research topic so far. Though there are few works on cooperative lane changing problem [19–21], none of them offers a generalized solution that can be applied in a scenario with any number of lanes or vehicles. This motivates us to introduce the above problem as a new line of research and propose a generalized algorithm for the same.

Lane Sorting is a term that we have coined for the collective process of rearranging a group of vehicles traveling initially on random source lanes such that each vehicle reaches its destination lane. The choice of destination lanes for a vehicle may depend on its velocity, destination group, size, tariff at a toll gate, priority, etc. Lane Sorting is a new concept in the ITS domain and it finds application in algorithms that manage vehicles in scenarios such as platoon formation, intersection management, toll-point execution, lane changes on highways, etc.

1.5.2 Autonomous Intersection Control

Intersection is a very crucial part of any traffic network. As they are the shared resource for all the incoming roads, intersection congestion can result in congestion in all of them. This is the reason, intersections are also known as the "Bottlenecks of traffic". Along with that, 40% of all traffic accidents and 20% of traffic-related fatalities in the U.S. involve intersections [22]. The economic loss caused by traffic congestion has been estimated to be over $ 88 billion in the U.S. in the year 2019 alone [2]. This suggests that the existing intersection management technique such as traffic lights, stop-signs, manual control, etc. are not up to the mark with the increasing levels of congestion and there is a need for a new approach for the same.

With the developments in the field of autonomous vehicles, a new paradigm of intersection management has been introduced. We observed that it is not required to have level-5 autonomous vehicles before we could take advantage of self-driving features in traffic management. For instance, level-2 vehicles can have autonomous control over their lateral as well as longitudinal movement; this qualifies them to be used in ITS applications requiring only these controls.

Motivated by our observations, we attempt the problem of intersection management of autonomous vehicles with a fixed velocity based heuristic approach. A fixed velocity-based approach results in no queuing before the intersection and it results in maximum economy,

minimum emissions, and maximum efficiency as it does not involve frequent acceleration and deceleration of vehicles. A heuristic algorithm does not involve heavy computational procedures thus making the algorithm suitable for real-time applications.

1.5.3 Formal Verification of ITS Algorithms

Owing to the safety-critical behavior of ITS scenarios, it is of critical importance to have a mechanism to obtain the correctness proof of an associated algorithm with respect to certain properties. Obtaining such a proof would involve modeling the scenario, vehicle behavior, modeling the concerned algorithm, and finally modeling the safety-critical requirements such as the no collision property. Besides, it is also crucial to reflect the real-world distributions in the model used for verification such as the spawning of vehicles. Finally, to check the sanity of the model itself, there should be internal verification routines to gain confidence in the verification results obtained.

1.6 Contributions

Here we present the significant contributions of the research work carried out for efficient management of autonomous vehicles at (i) scenarios that involve cooperative lane change maneuvers using the Cooperative Lane Sorting (CLS) algorithm and (ii) intersections using the Heuristic Autonomous Intersection Management (HAIM) algorithm. Before this, we coin the term Lane Sorting which refers to a line of research that has been overlooked so far. Thereafter, we propose a systematic procedure for the formal verification of the HAIM algorithm. A brief description of these contributions are described in the following:

1.6.1 Cooperative Lane Sorting

In this work, we introduce the concept of Lane Sorting. Thereafter, we propose the Cooperative Lane Sorting algorithm that performs the lane sorting operation on vehicle batches called frames. To perform lane sorting operation on a frame, an optimization formulation is performed which is then reduced to mixed-integer linear programming (MILP) formulation. The constraints designed in the MILP formulation make sure that all the frames are kept

independent from each other. This makes the CLS algorithm suitable for parallelization as each frame can be processed independently. The CLS algorithm is a cooperative algorithm in the sense that vehicles that are not currently attempting to change lanes will cooperate with lane changing vehicles and will make space for them. The CLS algorithm considers all possibilities regarding the existence of the solution to the MILP problem and ensures that the incoming traffic is eventually sorted. Additionally, CLS is a generalized algorithm in terms of number of lanes and vehicles wanting to change lanes, making it applicable for cooperative lane change operation of a group of vehicles as well as individual vehicles.

1.6.2 Heuristic Autonomous Intersection Management

In this work, a heuristic algorithm for intersection management of autonomous vehicle traffic is proposed named Heuristic Autonomous Intersection Management (HAIM) algorithm. The HAIM algorithm realizes a velocity-based algorithm in which vehicles incoming to the intersection first attain the assigned velocity and then maintain it throughout the rest of the traversal in the scenario. The HAIM algorithm consists of 4 levels of heuristics that successively resolve conflicts in the lane as well as intersection traversal of every vehicle incoming to the scenario. Being a heuristic algorithm, HAIM requires no computationally intensive procedure and has the complexity of the order of $\mathcal{O}(n)$, where n is the number of vehicles incoming to the intersection as every vehicle has to be considered once to calculate its velocity.

1.6.3 Formal Verification of the HAIM algorithm

In this work is presented a formal verification approach to prove that the HAIM algorithm resolves all conflicts and results in collision-free scheduling of vehicles. The presented methodology is unique in the sense that it makes use of a systematic technique for (i) formal verification of the HAIM algorithm (ii) verification of the HAIM model using internal verification, and artificial error injection technique. Later we perform simulation-based verification for ensuring faithful modeling of the HAIM algorithm.

Following are the publications corresponding to the contributions mentioned above.

- Aaditya Prakash Chouhan and Gourinath Banda, “Autonomous Intersection Management: A Heuristic Approach”, *IEEE-Access*, Vol. 6, pp. 53287–53295, 2018, DOI = "https://doi.org/10.1109/ACCESS.2018.2871337"

- Aaditya Prakash Chouhan, Gourinath Banda and Kanishkar Jothibasu, “A Cooperative Algorithm for Lane Sorting of Autonomous Vehicles”, *IEEE-Access*, Vol. 8, pp. 88759 - 88768, 2020,
  DOI = "https://doi.org/10.1109/ACCESS.2020.2993200"

- Aaditya Prakash Chouhan, Gourinath Banda, “Formal Verification of Heuristic Autonomous Intersection Management Using Statistical Model Checking”, *Sensors*, Vol. 20, 2020,
  DOI = "https://doi.org/10.3390/s20164506"

1.7 Organization of the Thesis

The rest of the thesis is organized as follows. Chapter 2 gives an introduction to autonomous vehicles and the underlying techniques for model-based formal verification which are Timed Automata (TA), Computational Tree Logic (CTL) and their probabilistic extension, and Statistical Model Checking (SMC). Chapter 3 describes the existing ITS algorithms related to autonomous intersection management and lane sorting. In Chapter 4, we first introduce the concept of lane sorting and then propose the CLS algorithm for the same. We then present simulation details and corresponding results. In Chapter 5 we propose the HAIM algorithm. Later in the chapter, we present the result of the comparative analysis with three other intersection management techniques with respect to average trip delay. Chapter 6 presents the existing work related to the formal verification of autonomous systems and associated algorithms. In Chapter 7, we present the formal verification of the HAIM algorithm using statistical model checking. The HAIM algorithm and the model of HAIM are both subjected to verification in this chapter. The verification results are presented later. Chapter 8 concludes our study in the domain of intelligent transportation system algorithms and their

formal verification. Finally, the potential future research directions in these areas are discussed.

Chapter 2

Introduction to Autonomous Vehicles and Formal Verification

This chapter provides an introduction to autonomous vehicles and underlying techniques in the applied formal verification methodology. We will discuss the evolution of autonomous vehicles, their classification, and their hardware and software architecture. Later we will discuss certain topics related to model-based formal verification such as timed automata, model checking, property specification, and their probabilistic extension. This chapter helps creating familiarity of readers with autonomous vehicles and model based formal verification and helps in creating a context of the works presented in this dissertation.

2.1 Autonomous Vehicles

2.1.1 A Brief History

The next revolution in transportation led by autonomous driving technology is just around the corner and in the very near future, we will be able to see widespread deployment of self-driven vehicles on public roads. The technology that drives this revolution is going to transform the way vehicles are operated. This transformation from manually driven to autonomously driven is currently the sole purpose of this revolution.

On average, humans are great drivers. It's surprising how easily and efficiently humans can perform driving tasks that includes performing environment perception, path planning, ve-

hicle control, etc. simultaneously. Also, some of the very complex dynamic driving tasks come naturally to humans especially when it involves perception, reasoning, and sensing. Indeed, the technology available today is not a match to some of these capabilities of humans. However, despite this, the data shows that humans are responsible for more than 90 percent of traffic-related accidents. The reason for such a poor turn out in numbers is that the performance of humans is not consistent. Humans are vulnerable to distractions, dizziness, emotional spurt, etc. Autonomous driving on the other hand is very consistent.

Though autonomous vehicles have attracted a lot of attention and development in the past decade, its idea has been there for a very long time. General Motors in Futurama exhibit at the 1939 New York World's fair predicted what the world would look like in 20 years and that prediction included an automated highway system with self-driven vehicles. In 1986, Ernst Dickmanns, a German scientist led a team to develop a self-driving car that could drive up to 60kmph. Later in the early 1990s, they worked on the Prometheus project under which an autonomous Daimler Benz drove from Munich to Copenhagen with traffic for a distance over 1600 kilometers having human intervention every 9 kilometers. In 1995, Carnegie Melon University's Navlab project completed the cross-country journey called "No Hands Across America" with more than 5000 kilometers of driving. The vehicle drove itself for 98.2% of this journey and for the rest of the journey, human intervention was required. In the year 2004, DARPA conducted its first Grand Challenge for autonomous vehicle developers throughout the world to participate and showcase their technology. This competition has played a pivotal role in the evolution of autonomous vehicles. Though none of the participating teams could finish the competition in the first version of this competition, it resulted in attracting serious and planned efforts in the development of autonomous vehicle technology. As a result, in the second version of the DARPA's grand challenge (2005), 5 teams out of 23 completed the race, and Stanley, the winning vehicle from Stanford University completed the race in minimum time. DARPA conducted its first competition in the urban setup in the year 2007 and called it the DARPA Urban Challenge. In the year 2012, Nevada became the first US state to allow the testing of autonomous vehicles on public roads. In 2016, nuTonomy taxis became operational in Singapore. By October 2018, Google autonomous car division now known as Waymo has driven a total of 10 million miles of autonomous driving.

2.1.2 Autonomous Vehicle Classification

Before continuing to the classification of autonomous vehicles, let us first discuss some of the key components of autonomous vehicles and some of the terms associated with their operation.

- Driving task: The driving task is a broad term and includes most of the operations that an autonomous vehicle performs to drive itself. It consists of three sub-tasks which are:

  - Environment Perception: In this sub-task, the vehicle is required to study its environment using various sensors and find out the static and dynamic obstacles that can include road curbs, other vehicles, cyclists, pedestrians, potholes, etc.
  - Motion planning: In this sub-task, the vehicle is required to plan its journey to travel from point A to point B. The motion planning sub-task can be further divided into long term motion planning and short term motion planning. Long term motion planning problems are generally of the form "Which route should be chosen to go from my current location to the central park?". On the other hand, short term motion planning problems are of the form "Is it safe to make the lane change to the right?"
  - Controlling the vehicle: In this sub-task, the main objective is to control the movement of the vehicle. This deals with the control of the actuators such as steering, brake, throttle such that the motion planning instructions are followed reliably. All control tasks can be classified into two categories which are lateral control, and longitudinal control. Lateral control is established using the steering of the vehicle and the longitudinal control is established by using throttle and brake.

- Object and Event Detection and Response (OEDR): It refers to the ability of an autonomous vehicle to detect an object or an event in the environment that needs imme-

diate response and then reacting to it appropriately.

- Operation Design Domain (ODD): This defines the operating conditions under which the autonomous vehicle is designed to function. This will include the weather conditions, time of day, type of roadway, and other characteristics of the scenario of operation of the vehicle. Defining the ODD of the vehicle is crucial for ensuring the safety of operation of the vehicle.

As the allied technologies improve, autonomous vehicles' capabilities are also monotonously increasing. However, they do have a certain level of autonomy with respect to conventional manual vehicles. This level of autonomy that a vehicle poses is based on the tasks that are performed by the autonomous driving system. The Society of Automotive Engineers (SAE) has given the classification of vehicles into 6 different levels of autonomy. Let us next discuss the SAE standard in detail and see how the level of autonomy for any vehicle is decided.

2.1.2.1 SAE Autonomy levels

The factors that define the level of autonomy of any autonomous vehicle are the following:

- Automated lateral control: Can the vehicle have autonomous control over its lateral movement in any or all of its driving modes?
- Automated longitudinal control: Can the vehicle have autonomous control over its longitudinal movement in any or all of its driving modes?
- OEDR: Can the vehicle perform all the object and event detection and can appropriately respond autonomously to them? Also, can the vehicle autonomously attain a safe state in an emergency situation?
- ODD: Does the operation of the vehicle is restricted to an ODD?

The following classification given by SAE, based on the factors defined above classifies autonomous vehicles into 6 different autonomy levels. We will be using the term Automated Driving System (ADS) as the actor whenever the given task is performed by the vehicle autonomously.

- **Level 0:** ***No automation***

  Level 0 autonomous vehicles are not autonomous vehicles rather they are manually driven vehicles without any advanced driver assist systems. All dynamic driving tasks such as perception or driving environment monitoring, steering, brake, throttle are done by the human driver. The task distribution for this level of autonomous vehicle is given in Table 2.1.

  Table 2.1: SAE Level-0 autonomous vehicle

Tasks	Done by
Lateral control	Human
Longitudinal control	Human
OEDR	Human
Fallback	Human

  Level 0 vehicles can only receive warnings from supervisory features such as lane departure monitoring but the control is at all times in the hand of the human driver.

- **Level 1:** ***Driver assistance***

  Driving assistance features that can control either latitudinal or longitudinal movement of the vehicle in one or more driving modes are present. The task distributions for this level of autonomous vehicle is given in Table 2.2.

  Table 2.2: Two possible task distributions for SAE Level-1 autonomous vehicle

Tasks	Done by
Lateral control	ADC
Longitudinal control	Human
OEDR	Human
Fallback	Human
ODD	Restricted
Example feature	Lane centering

Tasks	Done by
Lateral control	Human
Longitudinal control	ADC
OEDR	Human
Fallback	Human
ODD	Restricted
Example feature	Cruise control

- **Level 2:** ***Partial automation***

  Advanced driving assistance feature is present that controls both latitudinal and longitudinal movement of the vehicle in one or more driving modes. The task distribution for this level of autonomous vehicle is given in Table 2.3.

Table 2.3: SAE Level-2 autonomous vehicle

Tasks	**Done by**
Lateral control	ADC
Longitudinal control	ADC
OEDR	Human
Fallback	Human
ODD	Restricted
Example features	GM super cruise, Nissan pro-pilot assist

- **Level 3:** ***Conditional automation***

  All dynamic driving tasks are performed by the autonomous system of the vehicle in a supported environment. From this level, the autonomous operation of vehicles is not driving mode specific as the vehicle is responsible for evaluating the driving environment and performing all dynamic driving tasks. However, the human driver must take control in case of an emergency. The task distribution for this level of autonomous vehicle is given in Table 2.4.

Table 2.4: SAE Level-3 autonomous vehicle

Tasks	**Done by**
Lateral control	ADC
Longitudinal control	ADC
OEDR	ADC
Fallback	Human
ODD	Restricted
Example car	Audi A8 Sedan

- **Level 4: High automation**

  All dynamic driving tasks are performed by the autonomous system of car in a supported environment. Human driver is present but need not respond in case of an emergency; the autonomous system has fail safe strategies. The task distribution for this level of autonomous vehicle is given in Table 2.5.

- **Level 5:** ***Full automation***

  All dynamic driving tasks are performed by the autonomous system of car in all environments. There is no steering, and no levers for brake and throttle. The task distribu-

Table 2.5: SAE Level-4 autonomous vehicle

Tasks	Done by
Lateral control	ADC
Longitudinal control	ADC
OEDR	ADC
Fallback	ADC
ODD	Restricted
Example car	Waymo

tion for this level of autonomous vehicle is given in Table 2.6.

Table 2.6: SAE Level-5 autonomous vehicle

Tasks	Done by
Lateral control	ADC
Longitudinal control	ADC
OEDR	ADC
Fallback	ADC
ODD	Unrestricted

2.1.3 AV Hardware Architecture

We have seen the classification of autonomous vehicles depending on the functions performed by the autonomous driving system. Let us now see what are the hardware components that an autonomous vehicle uses to perform various tasks.

2.1.3.1 Perception hardware

Environment perception is done to get the knowledge of the environment and every object in it. It will include sensing static as well as dynamic objects. Through perception the autonomous vehicles will be able to make sense of the environment and itself and will be able to plan ahead its motion in the presence of all the obstacles. Typical static objects detected by perception module of an autonomous vehicle are:

- Road and lane markings
- Traffic lights
- Traffic signs

- Curbs
- Construction signs, obstructions, etc.

And some of the common dynamic objects are:

- Four-wheel vehicles
- Two-wheel vehicles such as Motorcycles, bikes, etc.
- Pedestrians
- Animals (Dogs, cats, ducks, etc.)

There are two classes of perception hardware components in an autonomous vehicle. The first is *Exteroceptive* and the other is *Proprioceptive*. The Exteroceptive perception devices are those devices which sense the presence of any object outside the vehicle for example, Camera, Lidar, Radar, etc. Whereas the proprioceptive perception devices sense the information of the vehicle itself, for example, Global Navigation Satellite System (GNSS), Inertial Measurement Unit (IMU) for sensing acceleration and heading, wheel odometry sensors, etc. This class of sensors are required for ego localization i.e. finding the position, velocity, orientation, and angular motion of the vehicle.

The environment perception is performed by the Exteroceptive sensors. The following are the list of such sensors commonly used on an autonomous vehicle.

- Camera: Camera is the most fundamental sensor that is required for environment perception. A camera is a passive device i.e. it requires an external source of light to capture images. The comparison metrics for a camera are its resolution, its field of view and its dynamic range. Resolution defines the quality of the image in terms of the number of pixels present in the image. Field of view as the name suggests is the span of the environment in terms of angle that the camera can capture. Dynamic range is the difference in the darkest and brightest tone that the camera can capture. A high dynamic range is critical for application in autonomous vehicle application to capture images in varying lighting conditions.
- LIDAR: Lidar stands for LIght Detection And Ranging, and is an active sensor i.e. it does not require an external source of light for its working. Lidar consists of several

light beams that reflect off the object and comes back to the detector. The time delay and difference in intensity of the reflected rays will determine the position of the reflecting object. Lidar gives a 3D scene geometry in the form of a point cloud. Mechanical lidar consists of a rotating element that has a stack of light beams which can be 8, 16, 32, or 64 in numbers. Higher the number of beams, higher is the resolution of the obtained point cloud. Since lidar has its own source of light, it is not affected by the external lighting conditions and can work equally well in dark as well as bright conditions.

- RADAR: Radar stands for RAdio Detection And Ranging and its make use of radio waves to detect any large object in the surrounding of the vehicle. Radars are very robust as they are least affected by precipitation and thus can work in adverse weather conditions. Comparison matrix of radar are detection range, field of view and position and speed accuracy. Radars typically come in two categories which are long-range with a short field of view and short-range with a wide field of view. Both these categories find applications in autonomous vehicles.

- SONAR: Sonar stands for SOund Navigation And Ranging and are used for short-range object detection. Sonar are quite inexpensive and are perfect for use in parking applications. Same as radar, sonar is also unaffected by the weather conditions such as fog and precipitation.

The proprioceptive or ego sensors that are commonly used in an autonomous vehicle are GNSS, IMU, and Wheel odometry sensors. GNSS are used to get the global position of vehicle along with its velocity and some times heading using satellite communication. IMU gives the acceleration, angular rotation, and orientation of the vehicle. Wheel odometry sensors gives the rotation rate and angle of the wheels of the vehicle. This information can be used to obtain velocity, heading, position, and orientation of vehicle.
Other than the sensor hardware, an autonomous vehicle will also rely on hardware components that are required for computational purposes. For that purpose, we need to have Graphical Processing Units (GPU), Field Programmable Gate Arrays (FPGA), and Application Specific Integrated Circuits (ASIC). Computing in an autonomous vehicle is a serious task and we need to have some high end processors to manage and process all the data that

are received from numerous external and ego sensors.

2.1.4 AV Software Architecture

The hardware components discussed above pass on a raw data in the form of numbers. For instance, an image returned from a camera is actually a two-dimensional array of pixel values and each pixel value is an array of three values corresponding to the intensities of the three channels present in a colored image. This raw data from the sensors has to be processed to make sense of it and to do that we need a software setup.
A possible software architecture of an autonomous vehicle has the following decomposition into five parts.

- Environment perception
- Environment mapping
- Motion planning
- Vehicle controller
- System supervisor

Environment perception is responsible for locating the ego vehicle in the space and detect and classify various surrounding objects from the raw sensor data. These objects can be static objects such as road markings, traffic signs, etc. or they can be dynamic objects such as other vehicles or pedestrians. Another classification that has to be made is what objects are on-road objects and what objects are off-road objects. This classification is crucial in the motion planning stage. Another way in which classification guides the motion planning is by deciding the amount of tolerance required for different classes of objects, for instance, a dynamic object classified as a pedestrian will have a more random nature as compared to an object classified as a car.
The environment mapping makes use of the objects detected in the environment perception stage to create maps for the motion planning stage. There are three kinds of maps that are created at the mapping stage. These maps are i) Occupancy grid map, ii) Localization map and iii) Detailed road map.

The motion planning stage is responsible for long-term and short-term motion planning of the autonomous vehicle. In long-term planning, trip planning and route planning is generally performed whereas, in the short-term planning, responses to the environment perception such as collision avoidance are planned.
The controller software is responsible for faithfully realizing the planned vehicle maneuver by controlling the actuators such as throttle, brake, and steering. Finally, the system supervisor supervises the complete software stack as well as the hardware components to make sure that everything is working as intended.

The technological developments and research efforts in developing autonomous vehicle technology has made the concept of autonomous driving very close to reality. At current time we have vehicles that can qualify to be level-4 autonomous vehicles. Though large scale adoption of autonomous vehicles by general public is not a reality right now, it is not far away in the future. We have seen how autonomous vehicles are dependent on robust and dependable technologies working together to make driving safer and more efficient. With the precision and accuracy of the autonomous driving technology, autonomous vehicles have a lot to offer to their owners as well as to the traffic management systems. As we have mentioned earlier, it is not necessary to have level-5 autonomous vehicles before we could exploit the advantages they have to offer. This suggests that the next wave of evolution of traffic control strategies is soon expected. We will present the proposed ITS algorithms in next two chapters but before that we will present the underlying techniques of formal verification in this chapter.

2.2 Formal Verification

Formal Verification (FV) is a systematic approach that uses mathematical reasoning to verify that the implementation (i.e. system model) satisfies the requirement/s (i.e. properties) [23]. The process of formal verification starts with the representation of the system using a formalism. This process of representing the system in a suitable formalism is known as *Modeling*. The outcome of modeling is a *Model*, which is a representation of the system or a part of the system under consideration. The choice of formalism to model the system depends most importantly on the type of the system to be modeled and the expressiveness required to model the key dynamics. Different types of systems that are possible are discrete systems, continuous-time systems, systems containing concurrent processes, hybrid systems, timed systems, probabilistic systems, etc. A reactive system with finite states without a notion of time can be modeled using Finite State Automata (FSA) [24] whereas, a time-critical system cannot be modeled using the same FSA. They will need a formalism that can either model time or temporal ordering among the states of the system.

Though there are various formalisms available for modeling and verifying real-time systems such as Promela [25], CSP [26], TLA+ [27], etc. Timed Automata (TA) is the most widely used formalism because it uses the most intuitive representation of states, transitions, clocks, and clock constraints associated with the system. Due to the wide acceptance of TA as the go-to formalism for real-time systems, it also has rich extensions in formalism for incorporating probabilistic/stochastic behaviors. Because of these advantages, we have chosen TA (actually Probabilistic TA, as we will see later) for modeling and verifying the HAIM algorithm.

Coming back to formal verification, there are broadly two classes of formal verification methods which are: i) Property-oriented verification and ii) Model-oriented verification [23]. In property oriented approach, the system is modeled using a set of properties that a system satisfies; that is, the system is represented using a set of mathematical equations. The required property from this system, which is also a mathematical equation should be a logical consequence of the equations that the system satisfies. Proving this logical consequence is the basis of verifying a system in property oriented approach. This procedure involves heavy use of natural deduction and proof methods using propositional and predicate logic.

Due to this reason, the property-oriented approach requires expert human guidance to obtain proof. The model-oriented approach on the other hand abstracts mathematical equations and uses the concept of states. The system is represented as a transition system, which is a tuple containing the possible set of states, transitions between them, and the set of properties each state satisfies. Verification in a model-oriented approach involves exploring the given transition system model for checking the satisfiability of the required properties on states. Algorithms available for doing this are automatic and do not need human intervention or guidance to obtain results. Added with the benefit of more intuitive development of system model using graphical editor, tools present for model-oriented verification offer a better choice for systems that can be represented using their underlying formal language. Due to these differences, model-oriented and property-oriented methods have different domains of applicability. The property-oriented method is more appropriate when we do not know what the system looks like and the best way to describe them is by the means of axioms. On the other hand, the model-oriented method is more appropriate when we do know what the model looks like and we can describe them rather precisely [28].

We will be performing formal verification of the HAIM algorithm, of which we are completely aware. We are also aware of the governing equations of motion of vehicles, the trajectories followed by these vehicles, and environmental details. Furthermore, we are not working with a system that is defined in terms of general properties and mathematical axioms. For these reasons, the model-oriented formal verification is more appropriate to formally verify the HAIM algorithm.

After the system has been modeled, we are next required to encode requirements in a requirement/property specification language. Examples of property specification languages for reactive systems are Liner-time Temporal Logic (LTL) [29], Computation Tree Logic (CTL) [30], regular expressions [31], etc. Usually, the choice of the property specification languages depends on the type of properties to be verified. For instance, to verify real-time properties over the system executions, a real-time temporal logic should be the right choice. We have used CTL as the property specification language as it allows us to write state as well as path formulas.

The choice of the modeling formalism and specification language coincides with that of the Uppaal model checker. Uppaal also requires the system model in the form of TA and the

requirements in the form of CTL formulas. Besides, Uppal allows the model checking of Probabilistic/Stochastic systems using the probabilistic extensions of TA and CTL in the variant Uppaal-SMC. During the verification of the HAIM algorithm, we make use of the Statistical Model Checking offered by the Uppaal-SMC model checker.

In the remainder of this chapter, we will present underlying techniques to build an understanding of the model checking procedure and its probabilistic extension. The topics that we discuss next are: i) Timed Automata, ii) Computation Tree Logic, iii) Model checking CTL iv) Statistical model checking, and v) Probabilistic CTL.

2.2.1 Timed Automata

We define here the notion of Timed Automata (TA) as defined by Alur and Dill in [32]. Let us consider $X = \{x_1, x_2, ..., x_n\}$ be a set of a finite number of real-valued *clock* variables. Valuation of these clock variables is defined over X by the map $\nu : X \rightarrow \mathbb{R}_+$ (non negative reals). The value of clock x_i can be represented using clock valuation as ν_i. Moving ahead, if $t \in \mathbb{R}_+$, $\nu + t$ will represent valuations given by $(\nu_1 + t,\ \nu_2 + t,\ ...,\ \nu_n + t)$. $[Y \leftarrow 0]\nu$ defines the valuation that assigns 0 to each clock $x \in Y$, where Y is an element in the powerset of X. Next, we define the set of clock constraints over X, $\Psi(X)$, where each constraint is of the form $x_i \prec c$ where $x_i \in X,\ c \in N$ and $\prec \in \{<, \leq, =, >, \geq\}$. If a clock valuation ν satisfies a clock constraint $g \in \Psi(x)$, we represent it as $\nu \models g$. This means that ν satisfies a clock constraint of the form $x_i \prec c$ whenever $\nu_i \prec c$. We next define the notion of time automata.

Definition: A Timed Automaton, $\mathcal{A}$, is a tuple $(L, X, Act, E, Label, guard, inv)$, where

(i) L is a finite set (non-empty) of locations with $l_0 \in L$ being the initial location,

(ii) X is a finite set of clocks,

(iii) Act is a finite set of actions,

(iv) $E \subseteq L \times Act \times \Psi(X) \times 2^X \times L$ is a finite set of edges,

(v) $Label : L \rightarrow 2^{AP}$ is a function that assigns to each location $l \in L$, a set of of atomic propositions valid in that location given by $Label(l)$.Here AP is the set of all atomic propositions,

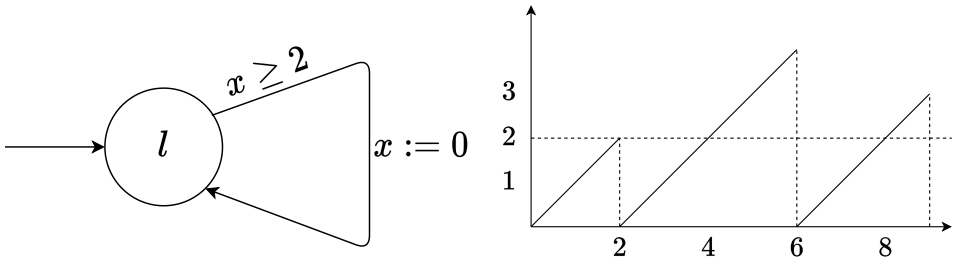
(vi) $guard : E \rightarrow \Psi(X)$ is a function that labels each edge $e \in E$ with the enabling condition given by $guard(e)$ over X, and

(vii) $inv : L \rightarrow \Psi(X)$ is the invariant function that assigns to each location, its corresponding invariant condition.

Before we take some examples of TA, let us first define the conventions used to draw a TA.

We will represent locations using circles and transitions between locations are represented using directed arrows. Invariants, unless *true* are specified within location circle. Edges representing transitions between locations are equipped with the enabling clock constraint by $guard(e)$ and the actions $Act(e)$. The labeling function will give the set of atomic propositions valid in a given location. These atomic propositions are specified over location circles. However, we will not be incorporating the labeling function in these examples because they are only required when we are concerned about model checking the given timed automata. Also, the initial location l_0 is represented using an incoming arrow.

Let us consider examples to understand the evolution of clocks and the meaning of various terms in timed automata.



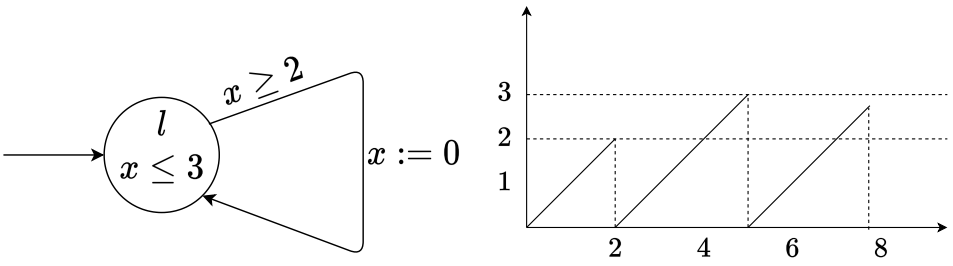
(a) (b) evolution of x with time

Figure 2.1: Timed automata example

Timed automata shown in Figures 2.1, 2.2, and 2.3 show single state automata to demonstrate the evolution of clock according to the guards, invariant and actions present. In Figure 2.1(a), the edge is enabled whenever $x \geq 2$ and every time the edge is traversed, x is reset to 0. As there is no invariant on the location, there is no time limit on the duration for which the TA can stay in the location l.

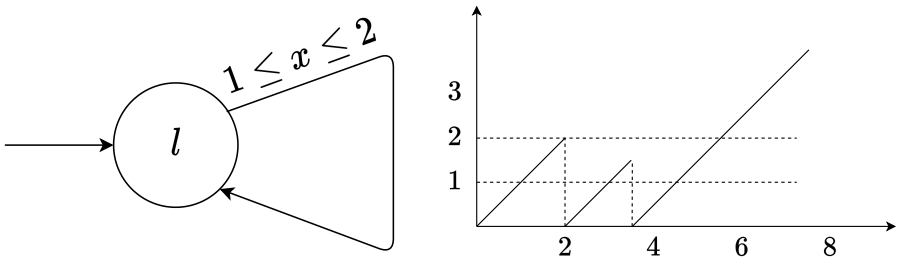
In Figure 2.2(a), there is an invariant at the location l thus restricting the time spent in the location. Since the edge is enabled only after $x \geq 2$, TA will be in location l for a duration of 2 to 3 time units each time.

Lastly, in Figure 2.3(a), there is no invariant at the location and the guard on the edge restricts passing through it only when $1 \leq x \leq 2$ and if this window is missed, this edge becomes disabled permanently.



(a) (b) evolution of x with time

Figure 2.2: Timed automata example



(a) (b) evolution of x with time

Figure 2.3: Timed automata example

2.2.1.1 Semantics of Timed Automata

A TA is interpreted in terms of an infinite transition system $(S, \rightarrow)$ where S is the set of states (pair of a location and clock valuation) and $\rightarrow$ is called transition relation that defines

how the transition system evolves from one state to the another. The transition system , $\mathcal{M}(\mathcal{A})$, associated with the time automata $\mathcal{A}$ is defined as $(S, s_0, \rightarrow)$ where,

(i) $S = \{(l, \nu) \in L \times \mathbb{R}_+^X \mid \nu \models inv(l)\}$. Here $\mathbb{R}_+^X$ is the set of all clock valuations over X,

(ii) $s_0 = (l_0, \nu_0)$, is the initial state of the transition system with $\nu_0 = 0$ for all $x \in X$, and

(iii) $\rightarrow$ is the transition relation given by $\rightarrow \subseteq S \times \Gamma \times S$ where Γ is an alphabet of actions.

The transition system of a TA is called finite if S and Γ are finite. In the above definition S is the set of states of the timed automata $\mathcal{A}$ and every state consists of a pair (l, ν) where $l \in L$ and $\nu \in \mathbb{R}_+^X$.
There are two types of transitions in a timed automata which are: *time-transitions* and *switch-transitions*. In a *time-transition* some positive amount of time is elapsed while staying in a location. This is allowed only when the invariant at that location is satisfied for the time of stay at that location. The time transition is defined as:

$$(l, \nu) \rightarrow (l, \nu + d) \text{ for } d \in \mathcal{R}_+ \text{ and } \forall d' \leq d, \nu + d' \models inv(l)$$

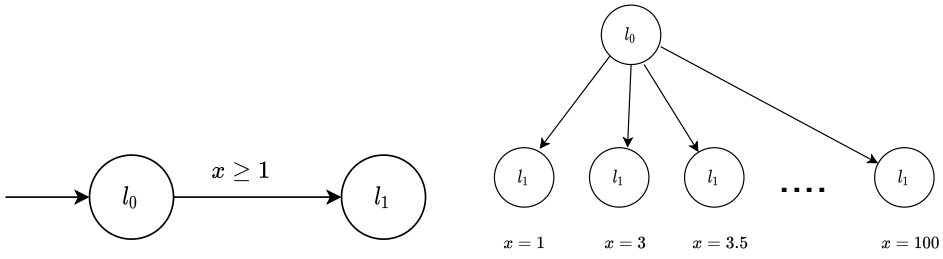
The second type of transition, the *switch-transition* involves two states $s = (l, \nu)$ and $s' = (l', \nu')$ of $\mathcal{A}$ such that there exists $g \in guard$ and $Y = 2^X$ such that $e = (l, g, Y, l') \in E, \nu \models g$ and $\nu' = [Y \leftarrow 0]\nu$. The switch transition is defined as $(l, \nu) \rightarrow (l', \nu')$ if:

(i) $e = (l, g, Y, l') \in E$,
(ii) $\nu \models g = guard(e)$ and
(iii) $\nu' = [Y \leftarrow 0]\nu$ where $Y \in 2^X$

Next, a *path* is defined as an infinite sequence of states $(s_i, i \in N)$ such that for all $i \in N$, the transition $s_i \rightarrow s_{i+1}$ can be performed either by a single *time-transition* or a single *switch-transition* or a pair of one *time-transition* and one *switch-transition*. The set of all the paths originating from the state s is denoted by $\mathcal{P}_m(s)$. This set is ranged over by the path variable, σ and $\sigma[i]$ will denote the $(i+1)^{th}$ state in that path.

2.2.1.2 Region graph

The verification of timed automata amounts to check whether $\mathcal{M}(\mathcal{A}), (s_0, \nu_0) \models \phi$; where ϕ is the specification required out of the system. However, the problem faced in checking this condition is that the state space of $\mathcal{M}(\mathcal{A})$ is infinite. See Figure 2.4 which shows a TA with two locations that will result into an infinite transition system.



(a) TA with 2 locations (b) Transition system of the TA

Figure 2.4: A TA and its corresponding transition system

To tackle this problem, the key idea by Alur and Dill [32] is to introduce an equivalence relation on clock valuations such that the same specification formulae are satisfied by equivalent clock valuations and there is a finite number of equivalence classes under this relation. We next define this equivalence relation between clock valuations and extend this relation to the states of $\mathcal{A}$.

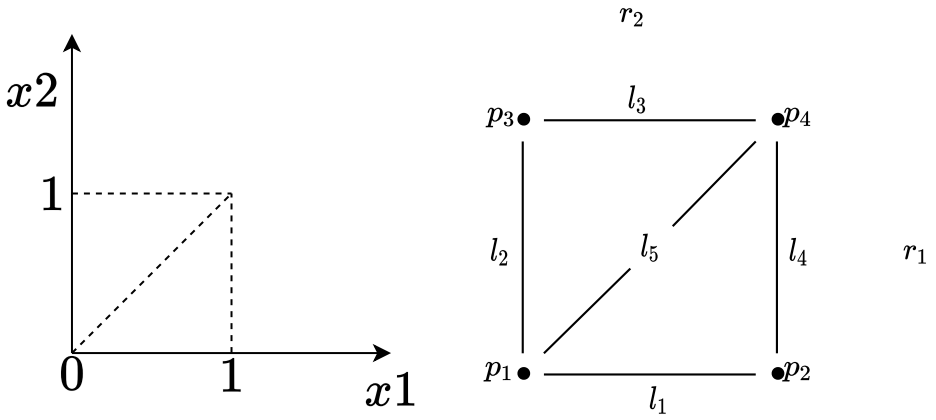
Let $\nu, \nu' \in \mathbb{R}_+^X$ and $M_{\mathcal{A}}$ be the maximum constant in guards of $\mathcal{A}$ then ν, ν' are said to be *region equivalent*, represented as $\nu \approx \nu'$ when the following conditions hold:

(i) $\forall x \in X, \lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ or $\nu(x), \nu'(x) > M_{\mathcal{A}}$

(ii) $\forall x \in X$ if $\nu(x) \leq M_{\mathcal{A}}, \{\nu(x)\} = 0$ if and only if $\{\nu'(x)\} = 0$, and

(iii) $\forall x, y \in X$, if $\nu(x), \nu(y) \leq M_{\mathcal{A}}, \{\nu(x) \leq \{\nu(y)\}$ if and only if $\{\nu'(x) \leq \nu'(y)\}$.

Here symbols $\lfloor\ \rfloor$ and $\{\}$ denote the floor and ceiling functions respectively.
This equivalence relation is now extended to determine equivalent states. Two states $s = (l, \nu)$ and $s' = (l', \nu')$ are equivalent if and only if $l = l'$ and $\nu \approx \nu'$.
Let us see how the state space of clocks of a TA is divided into finite number of region. Let us consider there are two clocks in a TA, x_1 and x_2. Also consider that this TA has the maximum constant in its guards as 1. The region automata will then consist of a total of 11 regions. Out of these 11 regions, 4 regions correspond to points, 5 to lines and two open areas corresponding to $x_1 > 1$ and $x_2 > 1$. These regions are shown in Figure 2.5.



(a) (b) Separate regions

Figure 2.5: Regions on the graph

All the regions in Figure 2.5(b) are as shown below.

p_1 : $x_1 = 0$ and $x_2 = 0$

p_2 : $x_1 = 1$ and $x_2 = 0$

p_3 : $x_1 = 0$ and $x_2 = 1$

p_4 : $x_1 = 1$ and $x_2 = 1$

l_1 : $0 < x_1 < 1$ and $x_2 = 0$

$l_2 :\ \ x_1 = 0$ and $0 < x_2 < 1$

$l_3 :\ \ 0 < x_1 < 1$ and $x_2 = 1$

$l_4 :\ \ x_1 = 1$ and $0 < x_2 < 1$

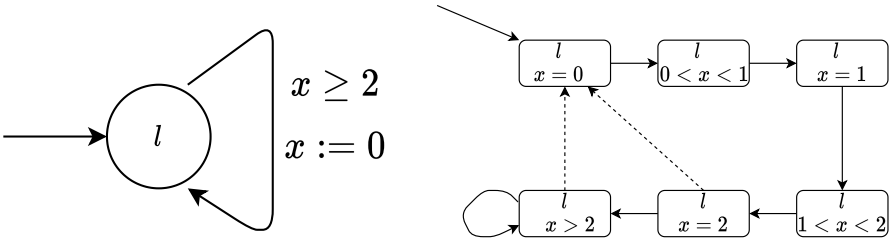
$l_1 :\ \ 0 < x_1 < 1$ and $0 < x_2 < 1$

$r_1 :\ \ x_1 > 1$

$r_2 :\ \ x_2 > 1$

In Figure 2.6, we can see how a timed automata can be approximated using a region automaton.

We have seen until now the notion of timed automata and we have also seen that an infinite



(a) (b) Region Automata

Figure 2.6: A TA and its corresponding Region automata

transition system may be generated from this time automata which is not suitable for performing exhaustive verification or model checking. We then described the concept of region equivalence in a timed automaton and discussed how it can be used to approximate a time automata with a region automata. Representing a system using timed automata and then converting the TA into region automata are the first two steps towards model checking the given system. For the next steps, we proceed with the property specification and then model checking the given properties. Next, we will discuss the CTL which is a branching-time logic for property specification.

2.2.2 Computational Tree Logic

Temporal logic is used to represent the ordering of events in time without explicitly introducing time. Temporal logic had been introduced for analyzing how time is used in natural

language arguments. Temporal logic is classified based on the notion of time used. In Linear Time Logic (LTL), a linear notion of time is used. Whereas in Computational Tree Logic (CTL), we have a branching notion of time.

LTL was introduced for specification and verification of systems by Pnueli in 1977 [29] and it has a linear evolution of the system with respect to time and at each time state (s), there is only one successor state and thus only one future (given by $R(s)$). This means that the evolution of the system with time can be represented with an infinite *sequence*: $s,\ R(s),\ R(R(s)),\ ....$

The computational tree logic was introduced by Clarke and Emerson in 1980 [30]. In CTL, states can have several different successor states and thus can have different possible futures. The underlying semantics thus have a notion of *tree* of states. In this tree, the sub tree rooted at a state s represents all possible transitions that start at s. The two notions of temporal logic discussed above have resulted in two streams of model checking procedures. We then discuss the syntax and semantics of CTL which is the basis of the formalism used in the UPPAAL model checker for the specification of properties.

2.2.2.1 Syntax of CTL

To define the syntax of CTL, we first define the set of atomic propositions. Propositions that can't be refined further in simpler propositions are called *atomic propositions*. Examples of some atomic propositions are: x is greater than 0, y is greater than 10, the cake is ready, etc. Let us call the set of atomic propositions AP, then the syntax of CTL is as follows:

(i) $\forall p \in AP,\ p$ is a CTL formula.

(ii) If ϕ is a CTL formula, then so is $\neg\phi$.

(iii) If ϕ and ψ are CTL formulas, then so are $\phi \vee \psi$ and $\phi \wedge \psi$

(iv) If ϕ and ψ are CTL formulas, then so are $AX\phi,\ EX\phi,\ AF\phi,\ EF\phi,\ AG\phi,\ EG\phi,$ $A[\phi U \psi],\ E[\phi U \psi]$.

Here A and E are path operators and X, F, G, and U are state operators. The above syntax in Backus-Naur form is given below:

$$\phi ::= p \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid AX\phi \mid EX\phi \mid AF\phi \mid EF\phi \mid AG\phi \mid EG\phi \mid A[\phi U \psi] \mid E[\phi U \psi]$$

The names and meaning of the used path and state operators are as follows:

- Path operators:
  - A : Along all paths
  - E : There exists at least one path
- State operators:
  - X : Next
  - F : Finally
  - G : Globally
  - U : Until

Let us take a few examples to understand CTL formulas better.
(i) The temporal formula $EG\phi$ will mean that starting from the current state, there exists at least one path in which ϕ is valid at every state. In short, $EG\phi$ can be pronounced as *Potentially always*.
(ii) $AX\phi$ will mean that for all paths, starting from the current state, the next state satisfies ϕ.
(iii) $AF\phi$: For all paths, ϕ is eventually satisfied at any state. (Eventually always).
(iv) $E[\phi U \psi]$: There exists a path in which ϕ is valid until ψ is valid.

2.2.2.2 Semantics of CTL

Since in CTL each state can have more than one successor states, we need to use the notion of *tree* to define the semantics of CTL. The CTL model is given by a triple:

$$\mathbb{M} = (S, R, Label), \text{ where,}$$

(i) S is a set of states

(ii) $R \subseteq S \times S$ is a relation on S that gives successor state at each state $s \in S$

(iii) $Label : S \rightarrow 2^{AP}$ labels each state with atomic propositions ($p \in AP$) that are valid in S.

The CTL model $\mathcal{M}$ is also known as *Kripke Structure*. This is somewhat similar to the transition system defined in timed automata.

Now we give the semantics of CTL in terms of satisfaction relation ($\models$) between the Kripke structure, $\mathcal{M}$, its states s, and the formula ϕ. We write $\mathcal{M}, s \models \phi$; read as "The Kripke structure, $\mathcal{M}$, at state s satisfies the property given by ϕ" if and only if ϕ is valid in the state s. Let $p \in AP$ is an atomic proposition and $\mathcal{M} = (S, R, Label)$ be a Kripke structure and ϕ and ψ be two well-formed formulas and let $\mathcal{P}_{\mathcal{M}}(s)$ denote the set of paths starting in the state s of the model $\mathcal{M}$. Then the satisfaction relation is defined as follows:

$s \models \top$

$s \not\models \bot$

$s \models p$ iff $p \in Label(s)$

$s \models \neg\phi$ iff $\neg(s \models \phi)$

$s \models \phi \vee \psi$ iff $s \models \phi \ \vee \ s \models \psi$

$s \models \phi \wedge \psi$ iff $s \models \phi \ \wedge \ s \models \psi$

$s \models (\phi \Rightarrow \psi)$ iff $s \models \neg\phi \vee s \models \psi$

$s \models AX\phi$ iff $\forall\sigma \in \mathcal{P}_{\mathcal{M}}(s)\sigma[1] \models \phi$

$s \models EX\phi$ iff $\exists\sigma \in \mathcal{P}_{\mathcal{M}}(s)\sigma[1] \models \phi$

$s \models AG\phi$ iff $\forall\sigma \in \mathcal{P}_{\mathcal{M}}(s)\forall i \geq 0, \sigma[i] \models \phi$

$s \models EG\phi$ iff $\exists\sigma \in \mathcal{P}_{\mathcal{M}}(s)\forall i \geq 0, \sigma[i] \models \phi$

$s \models AF\phi$ iff $\forall\sigma \in \mathcal{P}_{\mathcal{M}}(s)\exists i \geq 0, \sigma[i] \models \phi$

$s \models EF\phi$ iff $\exists\sigma \in \mathcal{P}_{\mathcal{M}}(s)\exists i \geq 0, \sigma[i] \models \phi$

$s \models A[\phi U\psi]$ iff $\forall\sigma \in \mathcal{P}_{\mathcal{M}}(s)(\exists j \geq 0\sigma[j] \models \psi) \wedge (\forall 0 \leq i < j\sigma[i] \models \phi)$

$s \models E[\phi U\psi]$ iff $\exists\sigma \in \mathcal{P}_{\mathcal{M}}(s)(\exists j \geq 0\sigma[j] \models \psi) \wedge (\forall 0 \leq i < j\sigma[i] \models \phi)$

Figures 2.7 and 2.8 show various examples depicting cases where each of the CTL temporal connective is valid. States filled with red color satisfy the condition ϕ and the ones with green color satisfy the condition ψ.

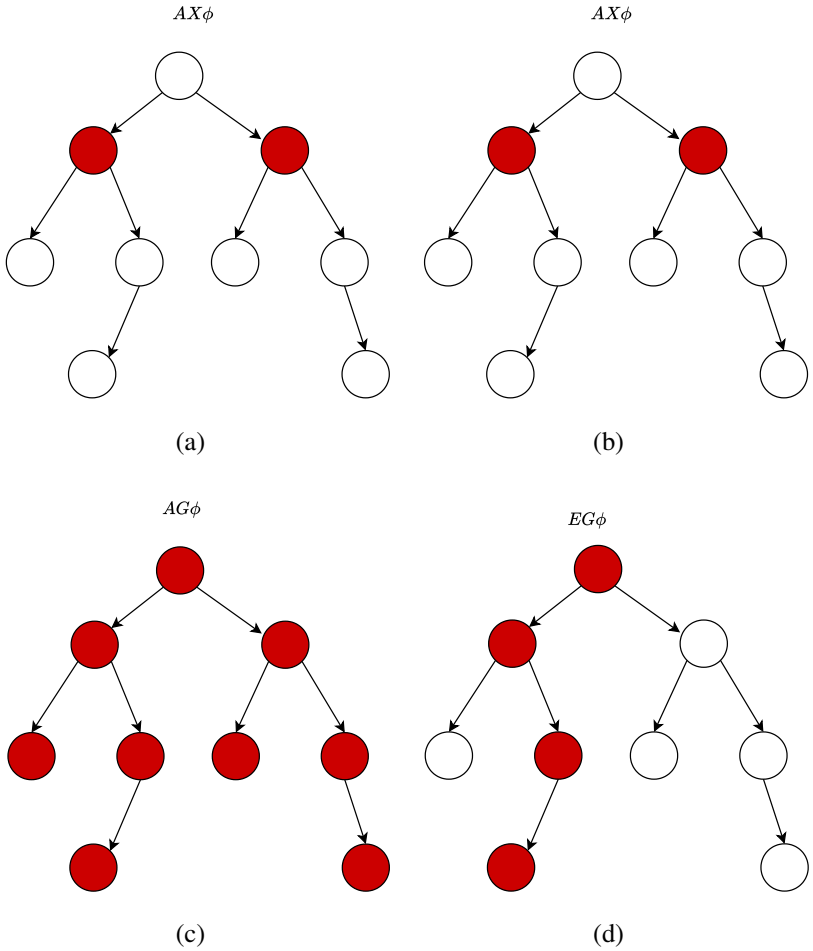


Figure 2.7: Examples for CTL formulas

2.2.2.3 Specifying properties in CTL

We will now see how properties are specified in CTL using an example scenario of process scheduling by an operating system. We will first write the specification in the language of discourse, then we will give its CTL translation.

Let there be two processes $P1$ and $P2$ that need to have a mutual exclusive access to a shared resource such as memory. Let us call this shared resource as Critical Section. The status of a request to operating system by a process p is represented by *RequestCS(p)*, the status of critical section occupancy by the process p is given by *inCS(p)* and the idle state of process p is given by *idle(p)*:

(1) Only one process can be in the critical section at any given time.

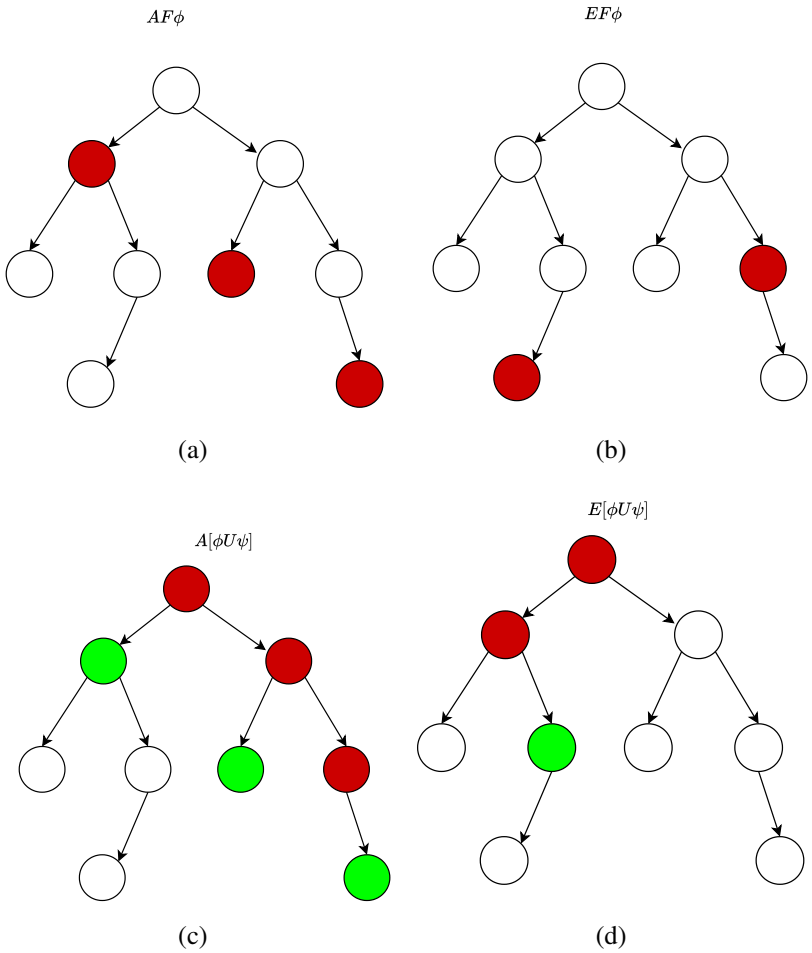


Figure 2.8: Examples for CTL formulas

$$AG\neg(CS[P_1] \wedge CS[P_2])$$

(2) a process will eventually enter the critical section whenever it wishes to do so.

$$AG(EnterCS[P] \;\Rightarrow\; AF\; CS[P]$$

(3) A process can always request to enter its critical section.

$$AG(Idle[P] \;\Rightarrow\; EX\; EnterCS[P]$$

2.2.2.4 Equivalence between CTL formulas

Two CTL formulas (say ϕ and ψ), are said to be semantically equivalent if for all states of a model at which ϕ is satisfied, ψ is also satisfied and vice versa. This equivalence is represented as

$$\phi \equiv \psi$$

Some of the equivalency relations between CTL formulas are given below.

$$\neg AF\phi \equiv EG\neg\phi$$
$$\neg EF\phi \equiv AG\neg\phi$$
$$\neg AX\phi \equiv EX\neg\phi$$
$$AF\phi \equiv A[\top U\phi]$$
$$EF\phi \equiv E[\top U\phi$$

Owing to these equivalences between CTL formulas, we can observe a redundancy among CTL temporal connectives i.e. one connective can be written in terms of some other connective. This fact enables us to choose an adequate set of connectives using which any CTL formula can be expressed. Using this adequate set of temporal connectives will result in a compact and simplified model checking procedure. In CTL, the adequate set of temporal connectives will be a set that contains at least one from $\{AX, EX\}$, at least one from $\{EG, AF, AU\}$, and EU.

2.2.3 Model checking CTL

Let us consider that the set $\{EX, EU, AF\}$ is taken as the adequate set of temporal connectives and $\top, \neg$ and $\wedge$ be the adequate set of propositional connectives. We are going to see the labeling approach towards model checking CTL. In this approach, the algorithm returns all the states of the given input Kripke structure that satisfies the CTL formula ϕ.

In this approach, the given CTL formula ϕ is first rewritten in terms of the adequate connectives using previously discussed equivalences in the pre-processing step and then starting with the satisfiability sets of the sub-formulas of ϕ, the satisfiability set of ϕ is built using the algorithm shown in the Algorithm 1. This pseudo-code shown in the Algorithm 1 uses the function SAT that takes as input the CTL formula ϕ and the model $\mathcal{M}$ and returns the set of

states where the formula ϕ is valid. In this pseudo-code equivalence between CTL formulas is used to reduce all formulas in terms of three sub-functions corresponding to SAT functions of the formulas EX, EU, and AF. The respective pseudo-codes for these SAT functions are given in Algorithms 2, 3, and 4, respectively.

Algorithm 1: function $SAT(\phi)$

Input: $\mathcal{M} = (S, \rightarrow, Label),\ \phi$
Output: set of states in $\mathcal{M}$ satisfying ϕ

```
1  begin
2    case
3      φ = ⊤ : return S
4      φ = ⊥ : return
5      φ = p ∈ AP : return {s ∈ S | φ ∈ Label(s)}
6      φ = ¬φ1 : return S − SAT(φ1)
7      φ = φ1 ∨ φ2 : return SAT(φ1) ∪ SAT(φ2)
8      φ = φ1 ∧ φ2 : return SAT(φ1) ∩ SAT(φ2)
9      φ = φ1 ⇒ φ2 : return SAT(¬φ1 ∨ φ2)
10     φ = AXφ1 : return SAT(¬EX¬φ1)
11     φ = EXφ1 : return SAT_EX(φ1)
12     φ = AGφ1 : return SAT(¬EF¬φ1)
13     φ = EGφ1 : return SAT(¬AF¬φ1)
14     φ = AFφ1 : return SAT_AF(φ1)
15     φ = EFφ1 : return SAT(E[⊤ ∪ φ1])
16     φ = A[φ1 ∪ φ2] : return SAT(¬(E[¬φ2 ∪ (¬φ1 ∧ ¬φ2)] ∨ EG¬φ2))
17     φ = E[φ1 ∪ φ2] : return SAT_EU(φ1, φ2)
18   end case
19 end function
```

Algorithm 2: function $SAT_{EX}(\phi)$

Input: $\mathcal{M} = (S, \rightarrow, Label),\ \phi$
Output: set of states in $\mathcal{M}$ satisfying $EX\phi$

```
1  local var X,Y
2  begin
3    X := SAT(φ);
4    Y := pre∃(X);
5  return Y
6  end
```

The pre denotes traveling backwards along the transition relation such that $pre_{\exists}(Y)$ returns a set of states that can make transition into Y. Whereas $pre_{\forall}(Y)$ returns a set of states that can only make transitions into Y [33].

Algorithm 3: function $SAT_{AF}(\phi)$

Input: $\mathcal{M} = (S, \rightarrow, Label),\ \phi$
Output: set of states in $\mathcal{M}$ satisfying $AF\phi$

```
1  local var X, Y
2  begin
3      X := S;
4      Y := SAT(φ);
5      repeat until X = Y
6          X := Y;
7          Y := Y U pre_∀(Y);
8      end
9      return Y
10 end
```

Algorithm 4: function $SAT_{EU}(\phi,\ \psi)$

Input: $\mathcal{M} = (S, \rightarrow, Label),\ \phi,\ \psi$
Output: set of states in $\mathcal{M}$ satisfying $EU(\phi,\ \psi)$

```
1  local var W, X, Y
2  begin
3      W := SAT(φ)
4      X := S;
5      Y := SAT(ψ);
6      repeat until X = Y
7          X := Y;
8          Y := Y U (W ∩ pre_∃(Y));
9      end
10     return Y
11 end
```

The region automaton developed in Section 2.5(a) can now be model checked using the above CTL model checking algorithm. So far, we have discussed the notion of Timed Automata, Region automata, the CTL property specification language, and how the region automata can be model checked against the properties specified in CTL.

2.2.4 Stochastic Timed Automata

Timed Automata is widely used for verifying real-time systems. In fact, it is one of the most studied models for this purpose. Due to its popularity, it has attracted attempts towards extending this formalism to model a broader class of behaviors. One such extension is Stochastic Timed Automata or STA in short. STA targets modeling both real-time and

randomized aspects of a system. Keeping in mind the stochastic nature of various real-world processes, we can say that STA results in a more faithful system in comparison to the models generated earlier by approximating the behavior according to formalism.
The semantics of STA is based on TA but with introduced stochastic nature in the delay at any location and discrete choices at any location. In other words, we get an STA when we associate each location of TA with distributions over delay and choice of transitions from that location. The notion of STA is defined as shown below.

Definition: A Stochastic Timed Automaton, $\mathcal{STA}$, is a tuple

$$(L, l_0, X, Act, E, Label, guard, inv, (\mu_q, p_q)_{q \in L \times R_+^X})$$

where

- $(L, l_0, X, Act, E, Label, guard, inv)$ is a Timed Automata.

- μ_s: Defined in terms of s, a state in the transition system of TA, given by $s = (l, \nu)$ where l is a location in TA and ν is the clock valuation, μ_q is a probability distribution over R_+ and it governs delay in location s.

- p_s: similar to μ_s, p_s is defined in terms of s and is a probability distribution over the set of edges that are enabled in s.

Let us give an intuitive introduction to how STA are build over a TA. Consider the TA shown in Figure 2.9. Let us call the edges going from l_0 to l_0 as e_1 and the one going from l_0 to l_1 as e_2. Also, let ν_1 and ν_2 are the clock valuations corresponding to the clocks x_1 and x_2 respectively. Now, let us first consider the state given by $(l, (\nu_1, \nu_2))$ = $(l_0, (0, 0))$ which is the initial state. At this initial state, both e_1 and e_2 are disabled so probabilities p_s assigns 0 to both these edges.

Now, e_1 will be enabled when $0 < x_1 \leq 1$ and e_2 will be enabled only when $x_2 \geq 2$. The set of time intervals for which there is an output edge from l_0 is $]0, 1] \cup \{2\}$. A distribution possible for these two transition is the combination of uniform distribution over the interval $]0, 1]$, given by $(\mathcal{U}(]0, 1]))$ and the transition at $\{2\}$ given by $\delta(2)$. In other words, there are half chances that there will be a transition on e_1 and the instant of this transition has a

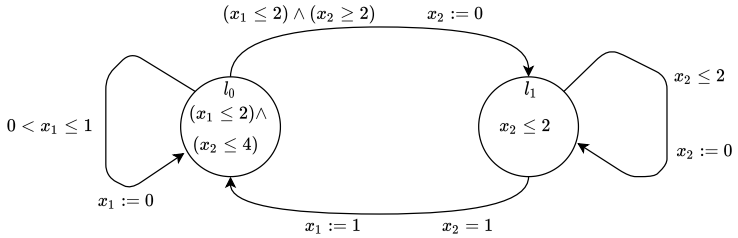


Figure 2.9: Example automata

uniform distribution and there is half chance that there will be a transition at edge e_2. Now let the state of the above TA is s_2 = $(l_0, (1.5, 3.5))$. In this state, e_1 is disabled and e_2 is enabled and also, the invariant of l_0 will be violated after 0.5 time units. This means e_2 has to be traversed by the time $x_2 = 4$. In this state the instant of transition for e_2 edge is given by the distribution $\mathcal{U}([0, 0.5])$.

Now consider the state $(l_0, (0.5, 3))$. At this state, edges e_1 and e_2 are both enabled. Edge e_1 will be disabled after $x_1 = 1$ and the invariant at l_0 will be violated after $x_2 = 4$. Therefore e_1 has a period of 0.5 time units. Thus the probability distribution for the instant of transition from location l_0 is given for e_1 and e_2 as: $(0.5 \times \mathcal{U}([0, 0.5]), 0.5 \times \mathcal{U}([0, 1]))$.

2.2.5 Statistical Model Checking

The statistical extension of timed automata that we discussed previously enables faithful modeling of real-world systems having inherent stochastic behavior. To perform verification of a stochastic system, one could approximate them with the nearest non-stochastic system and proceed with the methodology discussed so far. However, such verification will not be faithful.

Alternatively, the verification technique can be extended to work with the models consisting of stochastic behavior. The development of such verification algorithms is a non-trivial task as it requires capturing both the real-time as well as probabilistic verification.

To model-check stochastic system, one could make use of numerical techniques that iteratively compute the exact measure of paths satisfying given specification. The choice of technique to be used for obtaining the measure of paths depends on the stochastic system as well as logic used for specifying requirements. Though numerical techniques have made good strides, there remains various challenges. Numerical techniques require a lot of time

and space making them unsuitable for scaling to large systems. Besides, numerical techniques require the system to satisfy structural conditions specific to the numerical technique to be applied [34].

An alternative method to approach the verification of such stochastic systems is a simulation-based approach. The key idea in this approach is to determine the number of executions and then observe these many executions by a monitoring procedure. This monitoring procedure will infer using hypothesis testing, whether there is statistical evidence of the compliance of the given specification. Though the results provided by the simulation-based approach are not as accurate as of the numerical based approach, but the error in the result can be bounded to the desired accuracy by making a trade-off with the number of sample executions. The most significant advantage of the simulation-based approach is that it requires only a simulatable model of the system, thus increasing the class of systems it can be applied to. In addition to that, it can scale better to larger and more complex systems and it has far less memory and space requirements in comparison to numerical techniques.

2.2.6 Probabilistic Computational Tree Logic

We have seen the statistical extension to modeling (in STA) and verification (SMC) for systems having inherent stochastic nature. Along with these we also need to discuss the extension of the specification language to capture the stochastic behavior.

While dealing with a stochastic system we are not always concerned with qualitative properties only rather we may need to evaluate quantitative properties as well. This gives rise to two approaches towards model checking: Qualitative model checking and Quantitative model checking. Let us first discuss what is the difference between the two.

Qualitative model checking generally asks the question $(\mathcal{M}, s \models \phi)?$ i.e. whether the system (M) at state s satisfies the condition ϕ. The answer to such queries is either true or false. The model checking of real-time systems represented using timed automata that we discussed earlier falls under this category.

On the other hand, in quantitative model checking, the query is of the form $Prob(\mathcal{M}, s \models \phi)$ which means, what is the probability that the model $\mathcal{M}$ at state s will satisfy the condition ϕ. The result returned against this query will be an estimate of the probability of the compliance of the asked specification.

There are various systems and applications that require a quantitative answer in response to some query that encodes requirements on some important parameters for example cost and performance measures. Furthermore, there are systems for which the conventional qualitative model checking can't be used at their full potential as most of the various relevant properties are simply not valid in the model. This is commonly the case in systems involving probabilistic or stochastic behavior and one is forced to apply extensions of the conventional qualitative methods. We now discuss the syntax of the Probabilistic Computational Tree Logic (pCTL) and see how such quantitative requirements are encoded into specifications.

The grammar of pCTL in Backus-Naur form is given below

$$\phi ::= p \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \phi U \psi \mid \phi U^{\leq k} \psi \mid Pr_{\prec q}\{\phi\}$$

Here all the terms have their usual meaning as discussed in CTL syntax. Only except the probability operator Pr and the modified Until operator called *bounded until*. They are defined below.

- $Pr_{\prec q}\{\phi\}$ returns True if the probability that ϕ is True satisfies the relation given by $\prec$ where $\prec \in \{\leq, \geq, <, >\}$
- The bounded until, $\phi U^{\leq k} \psi$ returns true if ψ becomes true before k states and until that state, ϕ is true at every state.

Following are some example queries in pCTL

(1) Probability of not turning ON in the next step is at least 5%.

$$Pr_{\geq 0.95}\{\neg ON\}$$

(2) Probability of turning ON within 5 steps is at most 1%.

$$Pr_{\leq 0.01}\{\top U^{\leq 5} ON\}$$

2.3 Summary

In this chapter, we provided a gentle introduction to the underlying topics of this dissertation. In the first part of this chapter, we discussed the technologies that make a conventional vehicles an autonomous vehicle. Later, we discussed the classification of AVs. The attempt has been to create a familiarity with the autonomous vehicles for the readers. In the second part, we presented underlying techniques of model based formal verification and statistical model checking that helps in establishing an understanding of the technique applied in Chapter 7. In the next chapter, we give a survey on the literature associated to the two ITS scenarios considered in this dissertation.

Chapter 3

Intelligent Transportation Systems Survey

3.1 Lane Sorting Survey

The Lane Sorting scenario introduced in this dissertation involves the coordinated movement of vehicles to get each vehicle into its desired lane. In this scenario, vehicles are already aware of their destination lane based on which, they are classified as being part of different destination groups. This means the motive of lane change of vehicles in lane sorting scenario comes from the destination group they belong to and not from individual preference, which is common in lane-change maneuvers. Also, unlike lane-change planning which involves only those vehicles that wish to change lanes along with their surrounding vehicles, lane sorting scenario involves participation from all the vehicles in the scenario to accomplish the task. Furthermore, the lane sorting procedure is introduced with respect to some applications that may involve some deadline on the length of the road available to complete the task for example at tool-gates, intersections, etc. Despite the above-mentioned differences that distinguish lane sorting from available procedures involving lane changes of vehicles, there is some resemblance with the Cooperative Lane Change (CLC) procedure. This makes studying the literature associated with CLC crucial before presenting the proposed lane sorting algorithm. Along with the literature on CLC, we will also discuss some of the existing works on lane assignment and platooning of vehicles. Discussion on lane assignment tech-

niques is presented to investigate how vehicles are assigned destination groups to improve traffic efficiency. The discussion on platooning is also crucial due to the impact it can have on the efficiency, safety, and economy of traffic management and also because platooning attracts a direct application of the lane sorting procedure presented in this dissertation.

3.1.1 Lane Assignment

Lane assignment deals with the problem of assigning the most appropriate lane to any vehicle or a platoon. The objective of lane assignment is to improve the road capacity by minimizing the traffic disturbance activities such as lane change, overtaking, platoon splits, etc. In lane assignment the objective is generally to group vehicles with similar interests together in a lane. This grouping can be based on destination, size, average velocities, etc. of vehicles. The most common strategy is to group vehicles based on their destinations.

Lane assignment can be seen as a strategy that finds application prior to the platooning of vehicles. In platooning, vehicles arrange themselves into tightly spaced strings and the Lane Assignment will give the lane in which the platoon should be formed. Or we can also say that vehicles are assigned lanes by the lane assignment protocol and after vehicles arrive into their assigned lanes, they can later adjust their inter-spacing to form tightly spaced platoon. It is required to have platoons that remain intact for a longer period of time and also the platoons should have a large number of vehicles (roughly more than 5) to have major advantages.

In [18], authors propose strategies to organize vehicles into lanes at highway entrances such that the distance for which the platoon is kept intact is as large as possible. To give an idea of the need of lane assignment, they discuss that for an example scenario with 20 miles of average trip length and a platoon size of 5 vehicles, there is only 59% chances that platoon will stay intact between one exit and the other when a random grouping of vehicles is done. In [18], authors perform grouping by destination which turns out to be better than dynamic grouping. In [35], authors propose a distributed control strategy to assign the most appropriate lane to platoon of vehicles using inter-vehicle communication. This work is an extension to their previous work [36] in which they propose a lane assignment scheme for individual vehicles. In [35], authors formulate the platoon assignment for vehicles and lane assignment for platoons as linear programming problems with the objective of maximizing the traffic

throughput in the considered road segment.

Lane assignment and platooning both are crucial strategies for improving the throughput for any highway system. Lane assignment on one hand aims to decide the most appropriate lane for any given vehicle or a platoon such that there will be minimum number of traffic disruption events such as lane change, overtaking or lane reassignments. On the other hand, platooning will improve the capacity of road as vehicles will be tightly packed which will result in decreased fuel consumption due to reduced aerodynamic drag, and will result in more safe and comfortable journey for passengers. However to get the advantages of lane assignment and platooning, vehicles are required to come to the assigned lane from the lane it is currently traveling on. For a single vehicle, this will be a lane change operation however, the problem will change when there is a group of vehicles and every vehicle in that group has a destination lane which may or may not be same as its current lane. Suppose such a group of vehicle is entering at a highway entrance and a controller, as defined in [18] assigns destination lane to every vehicle. All these vehicles are required to come to their assigned lane and that too simultaneously. A lane change planning is itself considered a complex operation and here we are talking about collective lane changes of multiple vehicles. We call this problem as Lane Sorting.

Other than the scenario described above, we can find other scenarios where vehicles are required to be herded into lanes; for instance, at highway tolls where there are specified lanes for different types of vehicles, or at highways where vehicles are segregated in lanes according to their velocity.

3.1.2 Platooning

Platooning is a method that makes tightly spaced platoons of a group of autonomous vehicles that have common interests. Vehicles in a platoon follow the preceding vehicle and maintains a nearly constant distance with it. A platoon can be thought of as a train made of vehicles in place of compartments and these vehicles are connected to each other using wireless communication. Vehicles in a platoon can travel at high speeds with very small inter spacing resulting in an improved capacity of the roads. When vehicles travel in a platoon, there is a reduction in the aerodynamic drag which will result in better fuel economy and less exhaustion of gases. Platoon-based driving facilitates cooperative behavior of vehicles mak-

ing communication and control of a group of vehicles easier because instead of dealing with each vehicle in the platoon, only the platoon leader is to be dealt with. In a platoon based approach, using the advanced technology, driving can be made more safe and comfortable.
The primary objective of platooning is to improve the capacity of roads and reduce traffic congestion. Various programs have been organized taking into the consideration the potential benefits that can be realized by the application of platooning. The Automated Highway System (AHS) was developed at the University of California in association with California state department of transportation and the Federal Highway Administration under the PATH (Partners for Advanced Transportation Technology) program [8]. The main aim of the PATH program was to improve highway throughput by deploying platoons. The Grand Cooperative Driving Challenge [37], is an another program that aims to utilize the benefits of platooning on highways. Organized by the Netherlands Organization for Applied Scientific Research in 2011, the GCDC competition targeted cooperative driving on highways for a heterogeneous traffic. In heterogeneous traffic, vehicles having different size, technology, controlling algorithms, etc. are present and the participants were required to come up with strategies that does not require knowledge about other vehicles and were able to perform as good as possible. Another project called SARTRE [38], was a 3 year E.U. sponsored project started in 2009. The vision of this project was to develop and integrate strategies that enable vehicles to form platoons to reduce fuel consumption and reduce safety risks.
Other than the traffic optimization and reducing safety risks, applications of platooning include reduction of fuel consumption and gas emissions. The Energy ITS, [39], aimed at reduction of CO_2 emissions from vehicles as an application of communication technology. In an another work, [40], authors introduce the H_∞ method to increase highway capacity and decrease fuel consumption and emissions using well-organized platooning. The proposed control method is a multi-criteria control problems that considers fuel consumption, road inclination, emissions and traveling time of vehicles. Similarly in [41], authors propose a longitudinal control method for heavy trucks on highways to reduce fuel consumption.
Platoon based organization of the traffic is a complex task. It requires tight integration of the communication and control aspects. We will next see the literature associated with the management of vehicle platoons that involves three major tasks which are platoon formation, platoon split and keeping platoon stable.

In [42], authors give a scheme for platoon formation and split using a state transition diagram. The scheme is called V-PADA and makes use of the stochastic time series analysis to detect platoon formation, splitting and anomaly detection. In [43], authors propose a platoon formation scheme that allocates dynamic platoon-Id to vehicles using Filter Multi cast scheme. In [44], authors present the state machine of an application level protocol for the joint maneuver of a new vehicle not only at the last position but also at every position in the platoon. The stability of a platoon is defined in terms of the error in the inter-spacing of vehicles. This error is defined as the difference between the actual and the desired spacing between vehicles and it should always be bounded in time domain as well as frequency domain to ensure the string stability of the platoon. Stability of a platoon depends on factors such as parasitic lags and delays in the actuators of the vehicle, spacing policy in between vehicles, communication architecture, and control laws used. In [45] authors investigate the platoon stability of homogeneous and heterogeneous platoons with of adaptive cruise control enabled vehicles having constant time headway by considering parasitic time delays and time lags in sensors and actuators. They propose a sliding mode controller for the string stability of the platoon. In [46] authors perform a study of the influence of three time constants that affect the stability of platoons. These three time constraints are: reaction time of driver agent, velocity adaption time, and the numerical update time. They study the interplay of these time constants and how they affect the longitudinal dynamics stability of platoons of vehicles. In [47], authors propose a Safety Spacing Policy (SSP) as a superior strategy over the constant time gap policy for spacing of vehicles in platoon. SSP makes use of the current state of vehicle as well as the stopping capacity of vehicle to decide the spacing. In [48], authors propose a switching strategy that decides when to perform switching between the speed control mode and the headway control mode to satisfy constraints corresponding to passenger comfort, and collision avoidance with preceding vehicle.

3.1.3 Cooperative Lane Change

Sorting vehicles in lanes is a multi-vehicle lane change cooperation task. This will include motion planning of multiple vehicles which is inherently a difficult task due to the exponential scaling of computational complexity with the number of vehicles [19]. The work in [19] targets the collision free cooperative lane change of vehicles in a three lane scenario.

They consider eight vehicles because eight vehicles can cover all possible cases. They use a Minimum Safety Spacing (MSS) policy in which vehicles are required to have a safe spacing at the start of the lane change maneuver to guarantee no collision. This spacing policy along with other factors such as passenger comfort and efficiency of lane change are used to formulate the constraints for lane change trajectory calculation. This trajectory is calculated using optimization problem and the trajectory tracking is performed using Model Predictive Control strategy. In [20], authors propose an algorithm for solving online optimization problem for motion planning of vehicles during lane changes. They propose a Progressive Constrained Dynamic Optimization (PCDO) algorithm in which constraints of the motion control optimization problem are added progressively and the solution of the earlier stage is used as the initial point of the next stage. They also use a collision matrix that keeps track of colliding vehicles and the algorithm stops only when this collision matrix is null. The PCDO method facilitates forming lookup table for online application. In [21], authors have presented an algorithm to minimize the disruption of traffic flow and thus maximizing the number of lane changes. They used time slack calculation and the concept of vehicle grouping and used a distributed algorithm to solve the problem. [49] proposes lane change scheduling for vehicles on a two lane scenario. They consider a critical position earlier to which lane change operations should be completed. They first discuss lane change scheduling for a single vehicle and then extend this procedure for multiple vehicles so that all lane change operations are completed earlier to the critical position. They implement a rule based scheme that breaks down the problem space into seven cases and appropriate control scheme is given for each of these cases. In this work, for multiple vehicle lane changes, the single vehicle lane change procedure is repeated sequentially. Similarly in [50], one of the four predefined scenarios has to be attained to allow the lane change of vehicles.

In [51], authors propose a step wise computation framework to facilitate numerical solving process of the centralized multi vehicle motion planning optimization problem. They make use of an iterative process that excludes collision avoidance constraints at first and then sequentially adds these constraints until the original problem takes shape in the end. The objective function used targets to minimize the average time and the steering energy used by the group of vehicles. Directly applying the interior point method over the formed optimization problem would not result in a converging solution, as a result, the problem is divided into

multiple parts which are solved separately. In [52], authors present an analytical study performed on the traffics with different penetration rates of CAVs with an objective of studying how lane configurations and cooperative lane changing can affect the road way performance. They make use of the Intelligent Driver Model (IDM) to model vehicle behaviors. They also propose two algorithms for cooperative lane change operations corresponding to traffic with low penetration rates and traffic with high penetration rates. They perform a entropy based evaluation method to evaluate the performance of lane change algorithms. The work given in [53] proposes a decentralized decision making algorithm for cooperative lane changing of connected autonomous vehicles. The algorithm which consists of three modules, makes use of existing car-following models to predict vehicle's future states. An incentive based decision generation method is used in the candidate decision generation module. Then an algorithm is proposed to avoid candidate decisions that may result in conflict or traffic deterioration. In [54], authors propose a strategy to devise a politeness index that is used in making the lane assignment and lane change decisions by making a compromise between safety and efficiency. In [55], authors attempt to investigate optimal lane change behavior of CAVs by considering three lane change models. These three lane change models are (i) Minimizing Overall Braking Induced by Lane Changes (MOBIL) model [56] (ii) SUMO lane change model [57] and (iii) The one proposed in this work. The objective is to find how to optimize the cooperative lane change of CAVs and reduce the travels times, emissions and still letting vehicles to travel with their preferred speeds. In [58], authors propose a decentralized cooperative lane change framework for CAVs. The proposed algorithm makes use of a two-staged procedure in which the first stage corresponds to the arrangement of vehicles in longitudinal direction. The resulting arrangement is called as sparse formation. The second stage is where actual lane changes (lateral movement) are performed.

There are some works that are dedicated to lane change of individual vehicles with cooperative approach from other vehicles in the scenario. Some of such are reviewed next. In [59] and [60], authors defined the lane change trajectories as parameterized polynomials. While [59] using quintic polynomial for the purpose, and then applies nonlinear programming to solve for optimum safety, comfort and travel efficiency, [60] applied differential evolution algorithm. In [21] as well, sequential execution of lane change operation is performed for vehicles that are classified as belonging to different destination groups depending

on their destination lanes. In [61], the concept of lane change envelope is used which isolates the lane changing vehicle and as soon as this lane change envelope is not interfered by other vehicles, lane change can be performed.

3.2 AIM Survey

Efficient intersection management is the key to curb congestion. With the advent of connected and autonomous vehicles (CAVs), there comes way more possibilities to approach the intersection management problem. Wireless communication, revamped infrastructure, autonomous control of vehicles, in-vehicle sensors and processors, and applied intelligence drive the revolution in traffic management. These resources can be combined to establish traffic management in any scenario. In the literature related to the autonomous intersection control, as we will see, these resources have been used by researchers in numerous creative ways to achieve desired objectives. We will now be presenting a survey of literature associated with the autonomous intersection management. Although this literature is quite diverse in terms of adopted architecture, technique used, objective considered, approach used, etc., the most important classification comes from the used architecture of communication which can either be centralized or decentralized. There are very few articles that uses a mix of both, but we will talk about them separately.

The literature survey is presented below as consisting of two classes belonging to the two architectures of communication. First we will talk about the centralized scheme, in which all vehicle entities communicate with a central controller via wireless communication to send (ex. trip information, vehicle specifications, etc.) and receive (ex. reservation details, target velocity, etc.) information. In this class of solution, Vehicle to Infrastructure (V2I) wireless communication plays a crucial role. On the other hand, in decentralized communication, vehicle entities communicate within themselves to coordinate their access of the intersection area. The chief mode of wireless communication in this class of solution is Vehicle to Vehicle (V2V). Other than the classification based on the communication architecture, the literature can also be classified based on the objective, technique used, approach etc. We will be discussing these subcategories inside the main classifications.

3.2.1 Centralized Schemes

A majority of literature associated with the autonomous intersection control problem implements a centralized control policy in which a central controller entity makes control decisions

and communicate them to the vehicle entities. The central controller receive information from all the vehicles in the scenario and make the control policy in accordance with this information to ensure safety and efficiency in intersection operation.

3.2.1.1 Multi-agent based approach

A multi-agent system generally consists of various interconnected agents that are intelligent and/or autonomous. This approach is used in a large number of works to attempt the intersection management problem. We next review articles that use a multi-agent approach for solving the intersection management problem.

Authors in [62] present a centralized multi-agent scheme that uses reservation based scheduling of vehicles. They model the intersection as a grid of rectangular tiles. Vehicle agents communicate their trip details with the control agent called Intersection Manager (IM). In return, IM will reserve tiles, if available using First Come First Serve (FCFS) scheme for the vehicle agent. In case the reservation could not be made, the IM will send a denial message. The vehicle agent will need to resend the reservation request later. This scheme known as Autonomous Intersection Management (AIM) from the University of Texas at Austin can be regarded as the first significant research work targeting the control of intersection consisting of only autonomous vehicles. AIM later attracted various modifications and updates to its scheme. For instance, in [63], authors included driver agents that will have to make a turn at the intersection, included mixed traffic i.e. traffic with both autonomous and manually driven vehicles running simultaneously, emergency vehicles, varying granularity of rectangular grids and traffic density. They managed the mixed traffic using the modified reservation scheme which is called as FCFS-Light. In FCFS-light, manually driven vehicles can only pass through the intersection during green light whereas autonomous vehicles don't have this restriction and can pass through the intersection as dictated by the IM. Then in [64], authors proposed a motion controller that can plan the traversal of vehicle in the intersection region to increase its throughput. Taking inspiration from the Little's law in queuing theory [65], they proved using experimentation how optimized planning of arrival time and velocity of vehicles can result in increased throughput of the intersection. In [66], authors aim to enforce liveness into the intersection management. Liveness means that every vehicles who is waiting for a reservation at the intersection should eventually get one. This is not guaranteed

in [62] and could result in large inequalities in granting reservations for instance at the junction of a main street and an alley. Authors use batch processing of vehicles using added cost of waiting time of vehicles. In [67], authors propose an auction based autonomous intersection management scheme in which autonomous vehicles bid on behalf of their passenger for reservation at the intersection. Priority given in the reservation will depend on the bid made by the vehicle. For vehicles that have low budget, equality is established using the concept of system bids where the reservation manager bids on behalf of the vehicle. Experimentation results show that this scheme gives better results than the FCFS scheme. Then in [68], authors propose a Hybrid-AIM or H-AIM, that can schedule a mixed traffic at an intersection. The H-AIM strategy aims towards improved performance of the intersection scheduling even at 1% penetration rate.

Articles discussed so far have been derived or inspired from the actual AIM strategy given in [62]. The approach that most of these articles have is multi-agent based. Next we review some more articles that are modification of the AIM scheme and the ones that makes use of multi-agent approach to tackle the intersection management problem.

In [69], authors propose an interesting strategy inspired by economic approaches on computational markets to approach the intersection management problem. Here, vehicles, which are driver agents, have to trade with the intersection manager which acts as the trader and controls the allotment of the intersection space (resource). This trading is dependent on supply, demand and the pricing of the resource and follows the Walrasian auction [70]. Though this work is presented for any general traffic network, its applicability on an autonomous intersection management scenario is quite evident. Authors further explore the application of this strategy in their another article [71], where they present their market inspired approach as an alternative reservation policy to FCFS.

In [72], authors use the Game theoretic approach on a Multi Agent System (MAS) -based intersection system model. They use a heuristic optimization for finding the solutions. The basic approach in this work is that all vehicles in the scenario which are called Reactive Agents interact with the centralized Manager Agent in a game. Such action of each of the agent is chosen such that it results in minimum delay and safe evacuation of all vehicles through the intersection. In [73], authors propose a multi agent based approach that uses space-time slots in the intersections for making reservations. The Intersection Manager

Agent (IMA) incorporates the priority, order of vehicles in lane, and their order of arrival using three different reservation policies namely Priority-based, With-Lane-based, and FCFS policy. Their goals include increasing throughput, reduction in energy consumption and pollutant emissions. In [74], authors introduce an incremental data synchronization policy as an alternate communication protocol to the one followed in [62]. The main argument is that in AIM, when a driver agent's request for reservation gets rejected by the intersection manager, the driver agent will have to send new reservation requests later. These request packets will contain redundant data that are not time sensitive and does not change during a long period of time. They name the introduced policy as *ksync* and the results show that the average compression by each vehicle can be improved by over 80% when this policy is integrated into the knowledge base of the driver agents. In [75], authors propose a modification of the reservation policy used in AIM by introducing batch processing. In this work, authors use a mixed approach using both the FCFS policy as well as traffic light control. Noting that traffic light control works well for uncontrolled traffic, its advantages are used along with all the advantages of current ITS technology to develop an efficient algorithm in uncontrolled traffic situation as well. The work in [76] implements a multi agent representation of vehicles and intersection manager as Driver agents and Intersection Control System agent, respectively. Their objective is to utilize the intersection space in the best way along with resolving predicted conflicts by modifying trajectory of vehicles. For evaluating control action, a Fuzzy logic controller is used. The parameters of the output membership function of this fuzzy logic controller are optimized and tuned using the Genetic Algorithm. Simulation results show an improvement of 90.7%, 61.6%, and 72.4% in throughput, average delay time and maximum delay time, respectively.

3.2.1.2 Reservation based approach

In reservation based approach, the space and time are considered as resources which are shared by the incoming vehicles. To resolve conflicts, reservations of these space-time resources are made. We next review articles that implement this policy for intersection management of autonomous vehicles.

In [77], authors present a state-based approach to intersection management where a vehicle's intersection execution is modeled as a sequence of states. For these passing vehicles

reservations are made for critical section in accordance with their priority. They propose a state-controlled priority scheme called PriorFIFO. This scheme resulted in smaller delay for vehicles with higher priority as compared to FIFO scheme. Authors extend their work in [78], where they divide the intersection into continuous static critical sections which are reserved for vehicles. They propose a scheduling algorithm called *cs*PriorFIFO that considers the required QoS in deciding the priority. In [79], authors present an application of reservation based autonomous intersection management during emergency evacuation of cities or region.

3.2.1.3 Passing sequence based approach

In this approach, the sequence of vehicles is given the main consideration and the problem of efficient control of the intersection is performed by finding the optimum sequence of accessing intersection area by vehicles. We next review those articles that follow this approach.
In [80], authors propose a centralized dynamic programming algorithm to find the optimal passing sequence of vehicles that will evacuate all the vehicles into minimum time. Vehicles are first divided into Vehicle classes. Vehicles in one vehicle class can traverse through the intersection simultaneously if they are not in same lane. Vehicles in same lane have to follow the FIFO policy as overtaking in lane is not allowed. In [81], authors propose a meta-heuristic algorithm for finding the optimal passing sequence of vehicles at an intersection. First authors formulate the autonomous intersection management problem as a dynamic programming problem and find the complexity of finding the solution. They find that though the solution to the formulated problem using optimization is feasible but the real time performance for scenario with large number of vehicles and number of lanes will not be practical due to the exponential scaling of the solution space. They therefore applied the meta heuristic named Ant Colony system for the purpose. In [82], authors introduce an intersection management policy for autonomous vehicles called as Transparent Intersection Management (TIM). Authors consider the latency in communication and computation for their role in the real-time performance and attempt to keep the policy simple and efficient. They use the Intelligent Driver Model and rely on the fact that vehicles will obey the order given for its passage in the sequence decided by the server (Central controller). Policies used for deciding the sequence of vehicles at the intersection are First Come First Serve (FCFS)

and Distributed Clearing Policy (DCP). The DCP for autonomous vehicles makes use of the fact that vehicles in same lane have smaller headway time than those on conflicting lanes hence allowing maximum non-conflicting vehicles at the same time. In [83], authors don't specifically target autonomous vehicle traffic, rather is a general work that proposes a support system for vehicles at a signalized intersection. Often drivers are not sure what to do when they come across a transiting traffic light, wrong choice of action can result in either running the red light or a collision or unnecessary waiting time or a rear end collision. They devised a probabilistic sequential decision making process for such indecision zones prior to intersection. This decision making process makes use of decision rule which are either extracted from the existing parametric models of the indecision zones or created on the basis of data received from vehicle, intersection and traffic light. Though this work only requires connected vehicles instead of CAV, this can well be used in an ITS scenario consisting of CAV as well.

3.2.1.4 Optimization based approach

Optimization formulation always consists of writing two kinds of equation. The first one is the objective function which specifies the quantity that we have to maximize or minimize. And the other kind of equation that we write in any optimization formulation is constraints. Constraints formulate the limitations, restrictions on the state space of variables in the problem that should always be satisfied. In the context of autonomous intersection management, we generally come across optimization formulations that try to minimize the average delay in vehicle trip or try to maximize the intersection throughput as these are the most basic requirement from any intersection management protocol. There are some articles that also target minimizing jerks and velocity transitions to ensure passenger comfort or maximizing the Quality of Service or Quality of Experience of the passenger. On the other hand, constraints govern the state space of the variables of the optimization problem. In the context of the autonomous intersection management, constraints generally model physical limitations of vehicles such as limit on velocity, acceleration, jerk, and steering. They also formulate safety constraints such as "No two vehicles should be on same space at the same time".
Now we will review those articles that make use of some or the other optimization routine in a centralized fashion to find control actions.

In [84], authors propose a novel idea of adjusting the trajectories of vehicles to prevent collisions in the intersection. To implement this, they model the intersection management problem as a nonlinear constrained optimization problem in which the objective is to minimize the length of overlapping trajectories and constraints implement the limits on acceleration, velocity, and the minimum headway. For inevitable trajectory overlaps, infeasible solutions and system failure cases, a separate algorithm is given that ensures there are no collision in any case. They chose mobility measures (total travel time, total stop delay and maximum throughput) and sustainability measures (Carbon dioxide emission, Fuel consumption) as the measure of effectiveness of their proposed algorithm. They used the combination of VISSIM and MATLAB for simulation purpose. Their results show an improvement of 99%, 33% and 8% in the stop delay time, total travel time and throughput respectively. In [85], a model predictive control scheme is proposed. The objective of the scheme is to minimize the risk function where the risk function quantifies the risk of projected collision in terms of distance between the centers of vehicles crossing at the common conflict point. Authors performed the numerical simulation of this scheme and compared with the traditional traffic lights. Results show better traffic flow with the proposed scheme as compared with the traffic light. In [86], the author led a foundational theory for algorithm design that will guarantee the safety and liveness property in the coordination of ground traffic. They propose Model Predictive Control based formulations for lane follow, lane change and intersection control of vehicles. Protocols for inter-vehicle coordination are proposed to ensure safety and liveness in the autonomous ground traffic, examples for this are: Lane Change Protocol, Yield Protocol, and Intersection Control Protocol. In [87], authors propose a discretized model of intersection that consists of various nodes which are reserved for vehicles incoming to the intersection. The intersection is controlled by an Intersection manager which is communicated with the vehicles using the wireless communication (V2I). Vehicle is required to send its details to the intersection manager which in turn checks for availability of nodes in the trajectory of the vehicle. To calculate the trajectory of vehicles, Intersection manager uses dynamic programming. Results show that the given algorithm controls the intersection without any collision and with a decreased waiting time as compared to the traditional traffic light control. In [88], authors approach the collision avoidance in an autonomous intersection control algorithm using optimal control sub-problems for all permutations of crossing sequences of vehicles.

They formulate the problem as a convex program using space sampling rather than time sampling for finding optimal control sequence and implement the no-collision property as a constraint. Simulation results are shown for a case study of three vehicles to demonstrate the working of the algorithm. In [89], authors target a traffic of semi-autonomous vehicles that has communication capabilities. In this work, a supervisory routine is proposed that overrides controls when necessary to ensure safety. They formulate the supervisory control problem using mixed-integer programming approach. The advantages of this formulation is that it can handle complex intersections and due to reduced computation burden the scheme presents a better real-time applicability. In [90], authors divide the task of intersection management of autonomous vehicles into three sub-tasks. The first one is to find the limits of arrival times of each vehicle at the intersection, second is to solve for optimal schedules of these vehicles according to the limit provided by using the mixed integer linear programming (MILP). The third task is to find the feasible speed profile using which the vehicle can follow the given schedule without fail. Similarly, in [91], the intersection management problem is reduced to and solved using MILP. The proposed algorithm indirectly encourages vehicles to make platoon to further improve the throughput of the intersection. In [92] as well, authors propose the application of platoon to intersection management. Platoon based approach will result in reduced communication overhead and wait time at intersection. The technique used for scheduling of platoons at the intersection is a simple greedy algorithm. Authors in their other platoon based intersection control in [93], assign desired speed to vehicles when traveling towards the intersection and assign the acceleration and deceleration rates to vehicles that have conflicting trajectory. The model for vehicle movement restrictions and collision avoidance along with the objective to minimize the variance of velocity, results into a challenging nonlinear problem. The formulation is linearized into an MILP. The proposed algorithm succeeds in finding near optimal trajectories for vehicles without any collisions. In [94], authors propose a supervisory control routine using a model predictive controller for optimal sequencing of vehicles at the intersection. The intersection is modeled as a queuing system having a hybrid nature with integer queue length and continuous arrival/departure times. Results show that there is a better transient response when traffic changes and lower average delay in the trip times. In [95], authors propose an optimization based technique called Mixed integer programming based Intersection Coordination Algorithm (MICA) to

find the trajectory that vehicles have to follow while crossing the intersection. A vehicle in this scheme is either a following vehicle or heading vehicle. The intersection manager only deals with head vehicles to reduce its computation load. In [96], authors propose the Collision Aware Resource Allocation (CARA) strategy. In this work, to limit the communication channel congestion, only those vehicles are assigned the channel that are in a risk of future collision. The strategy implies a collision aware proactive strategy based on a Collision Probability Indicator (CPI) that calculates the probability of vehicle to end up in a collision. The traffic controller makes use of a Model Predictive Controller for optimal scheduling of vehicles at the intersection. By varying the threshold for CPI, control performance can be traded with the communication load. In [97], authors propose a game theoretic approach for intersection management that has game-in-game framework. The intersection management task is broken into two games. In the first game, vehicles participate in a platoon formation game and in the second game, collisions at the intersection are avoided. To maximize the intersection throughput, the problem is formulated using a linear optimization problem.

3.2.1.5 Miscellaneous

In [98], authors, with an aim to reduce delay, emissions and fuel consumption, present a cooperative system that performs traffic signal and vehicle maneuver optimization. For this purpose, authors use an architecture in which vehicles, that are all autonomous, transmit their local sensor information to a traffic signal controller using V2I communication. Using the information received from vehicles, this controller extract the tailback information and the overall traffic density information that it uses to generate optimized traffic signal plan. Along with the signal optimization, vehicle driving strategy is also optimized for both longitudinal as well as latitudinal control. This optimization devises the driving strategy that will result in minimum stop and lane change operations. To keep the computation time of the optimization procedure within real-time boundations, the Max-Min Ant System (MMAS) meta-heuristic is used. Simulation results shows that the presented system can reduce the number of required stops by 25%.

3.2.2 Decentralized Schemes

Decentralized control is another architecture that involves no central controller rather, the decisions are taken collectively/individually by the vehicles. Vehicles perform wireless Vehicle-to-Vehicle (V2V) communication to share information required to make control actions. In this section, we are going to review articles that make use of such architecture in their intersection management algorithm.

In [99], authors propose a novel technique for coordination of autonomous vehicles at the intersection. They introduce decentralized navigation function for vehicles with fixed paths. Navigation functions are the functions of position, velocity, acceleration or time which is used to define the trajectory of a robot such that the required condition such as collision free, or minimum time or minimum energy, etc. is met. They define the navigation function as a function of path, taking the vehicle inertia into account and also allows to define priority for some vehicles to save energy; for instance, large vehicles that have large associated inertia. It is shown that collision freedom is also guaranteed with the choice of the navigation function elected. Results show that the proposed method show better performance in terms of energy consumption in comparison to the traditional traffic lights and previously defined navigation function methods. In [100], authors discuss how decentralized approach brings robustness in the overall system and makes the technology more appropriate for applications such as search and rescue, hazardous site investigation and establishment of ad hoc communication. In this work, authors propose to have a decentralized approach for finding the solution to the AIM problem of [62] such that vehicles can within themselves find safe reservations without any centralized controller. In [101], authors present a scheme that is different from all the previous approaches. In this work, the shortcomings or disadvantages of centralized control are highlighted and then a distributed agent based framework is introduced that implements a self-organizing and cooperative control of CAVs. They point out the fact that FEFS scheme and competition between the driver agents may result in sub-optimal performance. The proposed scheme has three cooperation layers which are: Network, Route, and Intersection. At the network and route layer, information such as high level network information, route information for forming platoons, etc. are decided. However, this work specifically targets to determine the right-of-way for vehicles at the intersection. Each vehicle in the scenario

is given a priority level derived from the priority queuing principles using the information such as vehicle occupancy, vehicle type, vehicle's dynamics, etc. Each vehicle can calculate its own and receive other vehicle's priority level and thus determine the right of way at the intersection. Simulation was performed using the VISSIM simulator. Results showed decreased delay for all users. In [102], authors propose a rule set that can be used to define the right of way and thus find the sequence of vehicles to pass through the intersection. This rule set is designed based on the law of road traffic safety. For vehicles that need to decelerate in order to yield, an algorithm is also given to find the proper deceleration value. In [103], which is further extended in [104], authors propose a two-level system which enables autonomous vehicles to cross the intersection without stopping. The two levels in this work has objective of improving the efficiency of individual intersections and the overall network respectively. This work talks about "Green waves", which is a phenomenon in which most of the vehicles pass through the network without stopping. Synchronized traffic lights is a common approach to realize this. Along with forming green waves, this work focuses on two objectives which are minimizing travel times and minimizing the total energy consumption. In [105], authors propose an Intersection Control technique that can perform collision free scheduling of vehicles even in the presence of an unknown number of communication failures. A decentralized control is established which is later verified for safety and liveness properties. In [106], authors propose an improved Eco-Approach and Departure (EAD) algorithm in which instead of vehicle-centric approach, a cluster based approach is used. Vehicles are first assigned to clusters and then rearranged inside clusters. For each cluster, a leader is chosen and then passage through the intersection is decided using the EAD approach. The cluster based approach instead of vehicle based will result in 50% improvement in the throughput, 11% reduction in energy consumption and 20% reduction in emissions. In [107], authors propose a distributed algorithm for intersection management of vehicles in absence of any central controller. Vehicles share their expected schedule of accessing the critical areas of the intersection with all the neighboring vehicles. This information-sharing will aggregate the expectations and adjustments of vehicles in neighborhood. The aggregation of these information will lead to a conflict free schedule of all vehicles. The decision maker sub-module finds this schedule which is used by the motion planner sub-module. Since the schedules are decided based on the neighboring vehicles, the solution reached is

not a globally optimum solution.

3.2.3 Hybrid Schemes

In this section we will discuss those articles that propose solutions integrating both the centralized and decentralized control. In [108], authors propose both centralized as well as decentralized approach for controlling an intersection with two conflicting lanes. The intersection is modeled using timed Petri-Net and the objective of both these approaches is to vacate the vehicles in the intersection in minimum time by finding the best intersection access order of vehicles. In [109], authors approach the vehicle sequencing problem for four adjacent intersections.. They first divide the traffic on every intersection into different Compatible Stream Groups (CSG) which are groups of lanes that don't overlap in the intersection. Vehicles in each CSG are then classified into Passing Groups (PGs) and Fundamental mini Groups (FGs). The passing sequence of the FGs are then optimized using the Genetic Algorithm with an objective to minimize the Overall Evacuation Time. [110] presents a reservation based cooperative traffic scheduling mechanism for both centralized and decentralized architectures. All vehicle cooperative behaviors are modeled using event-triggered automata using which the cooperation mechanisms are defined that realize Reserve Advance Act Later (RAAL) procedure. Traffic manager agent uses special policies namely First-Arrive-First-Pass (FAFP), High-QoS-In-Prior (HQIP), and Longest-Queue-In-Prior (LQIP). This work is specifically not for intersection management rather for various scenarios such as lane change and overtaking. They use static critical section in case of intersection management whereas dynamic critical section in case of overtaking or lane change. [111] presents a vehicle routing approach that uses arrival on time and total travel time as objectives simultaneously. A semi-decentralized multi-agent architecture in which infrastructure agents perform the route assignment problem. To maximize the probability of reaching the destination before the deadline, probability tail model is used. This model has been used for various deadline sensitive emergency applications. To reduce the computational overhead the route assignment problem is formulated as an MILP problem. The influence of each vehicles on other vehicles is taken into account by collecting intention of all vehicles by the infrastructure agent. Route guidance procedure makes use of these intentions to perform route assignment. In [112], authors approach

the management of a group of intersections by having two scopes which are i) Grid management, and ii) Vehicle routing. For grid management they used the Advanced Cooperative Vehicle-Actuator System (ACVAS) which is an improvement over the CVAS defined in [113]. ACVAS does not optimize any particular intersection as it can result in congestion rather attempts to keep the grid deadlock free. For vehicle routing they use three weighting policies namely Distance, Reservation, and Congestion. These weighting policies weights the arcs in the network graph. In [114], authors propose a hybrid scheme of intersection control. A job scheduler acts as the centralized planner and assigns crossing schedules to vehicles. Whereas a distributed controller ensures that vehicles meet their schedule along with guaranteeing the no collision. In [115], authors attempt to solve the intersection management problem while considering the whole network instead of just an isolated intersection. They took a hybrid approach in which an isolated intersection is approached with centralized approach and the network in a decentralized approach. Different intersections communicate within themselves thus introducing Infrastructure to Infrastructure (I2I) communication. The network is represented using graph and intersections as nodes in the graph. The technique used is called the discrete consensus algorithm.

3.3 Summary

In the first part of this chapter, we discussed the existing literature related to the lane sorting procedure and algorithm. The works discussed here involve lane changes of vehicles using different cooperative algorithms. Most of these works are dedicated to lane change problems for individual vehicles and some permit lane changing of multiple vehicles but with sequential executions of individual lane change operations. Due to this limitation, there is sub optimal exploitation of the available resources. Furthermore, a majority of the works presented above keep the lane changing task only confined to the vehicles that are attempting to change lanes or are neighbors to those vehicles. The true cooperative lane change planning will be the one that involves all the vehicles in the scenario. Most importantly, the works discussed above are dedicated to the lane change operations that are primarily motivated by personal preferences such as high traveling velocity. In this way most of considered works

don't target lane change operation with an objective of arranging the vehicles in a more favorable organization of vehicles for efficient management of any traffic scenario. For instance, we know how platoons can make traffic management at various scenarios very efficient and economic but there is no algorithm to the best of our knowledge that attempts to cooperatively rearrange vehicles such that every vehicle reach its destination lane from it initial lane. Along with the application to platooning, the availability of such an algorithm will also be useful in arranging vehicles at scenarios that can be benefited by favorable arrangement of vehicles in lanes for example toll-gates, highway entrance and exits, intersections etc.

In the second part of this chapter, we presented existing work that address the autonomous intersection management problem. We observed, there are several significant features associated with each of these solutions such as safety, efficiency in terms of delay, throughput, emissions etc. with a hard condition on safety. A majority of works on autonomous intersection management adopts a centralized approach and various techniques used for the purpose are optimization, control-theory, market-inspired, naturally inspired, etc. Out of all the schemes, the fixed-velocity based scheme works on various objective at the same time. It results in no queuing at the intersection, minimum fuel usage and exhausts due to minimized acceleration and deceleration, and maximum comfort. If such a scheme could be developed with procedure involving low computational needs resulting in safe intersection traversal, then it would turn out to be a very efficient intersection management strategy.

Based on the above mentioned observations, we will first present the proposed algorithm for lane sorting in Chapter 4 and the Heuristic Autonomous Intersection Management algorithm in Chapter 5.

Chapter 4

Cooperative Lane Sorting

4.1 Introduction

Lane change is one of the most common maneuvers in driving. It is performed while overtaking, merging, diverging, pull-overs, etc. Having sound algorithms for planning lane change maneuvers is thus critical in terms of safety as well as efficiency in various scenarios. The objective of any lane change maneuver is to get the vehicle from its current lane to the destination lane safely and efficiently. Due to their importance in various driving scenarios, there are numerous works in literature dedicated to planning safe and efficient lane change maneuvers for a vehicle wanting to change lanes. Lane change planning such as lane change scheduling, maneuver planning, etc. for individual vehicle lane change operations has become an established research problem with various related algorithms.

There are various scenarios where a group of vehicles rather than an individual vehicle have to collectively plan their lane change operations for instance at toll gates where all incoming vehicles are required to be sorted in lanes corresponding to vehicle size and type, at freeways, there are dedicated lanes for faster moving vehicles which define a distribution of vehicles on lanes based on their velocity. Such scenarios require coordinated lane change operations of multiple vehicles sometimes added with some additional constraints such as a limit on the available road length to complete all lane change operations. The work presented in this chapter caters to the same situation where lane change planning for a group of vehicles is to be performed. Keeping such scenarios in mind, we next present a new ITS procedure known as Lane Sorting.

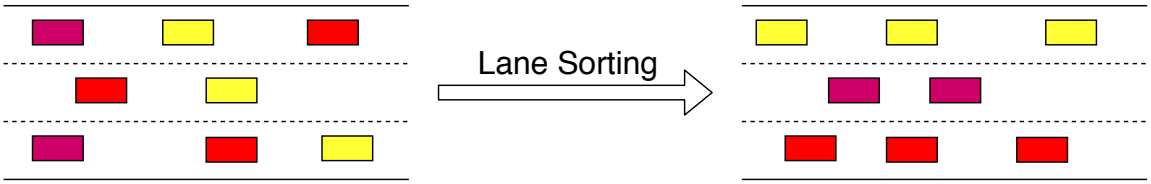


Figure 4.1: Lane Sorting Process

4.1.1 Lane Sorting

Consider a group of vehicles $(V_1, V_2, ..., V_m)$ travelling on a long stretch of straight road with n number of lanes $(l_1, l_2, ..., l_n)$.

Let us define the relation $CL(V_i)$ that gives the lane in which vehicle V_i is currently travelling and the relation $DL(V_i)$, that gives the destination lane or the lane in which the concerned vehicle wishes to move to eventually. Let us assume that the every vehicle is already aware of its destination lane.

Let us consider that case in which some of the vehicles in the set $(V_1, V_2, ..., V_m)$, are not travelling in there destination lane. For such a traffic setting, we could write:

$$\exists V_i \in (V_1, V_2, ...V_m) \quad \textbf{s.t.} \quad CL(V_i) \neq DL(V_i) \tag{4.1}$$

Then the lane sorting operation is defined as the procedure that converts, in a finite amount of time, the traffic setting given in equation 4.1 into a setting given by:

$$\forall V_i \in (V_1, V_2, ...V_m) \quad CL(V_i) = DL(V_i) \tag{4.2}$$

In other words, the objective of the lane sorting opertion is to physically rearrange vehicles to bring them all in their destination lane. The lane sorting operation is in contrast to the lane assignment technique as lane sorting manages the physical movement of vehicles in between lanes. On the other hand, the objective of lane assignment, as discussed in Section 3.1.1 is only to assign the destination lane to a given vehicle.

The diagram shown in Figure 4.1 gives a graphical representation of the lane sorting procedure.

Please note that it is not necessary to have as many lanes as there are destination groups of vehicles i.e. there can be a case where the number of lanes can be greater than or smaller than the number of destination groups. In case the number of destination groups is greater than the number of available lanes, some of the lanes are shared for multiple destination groups, and in case the number of destination groups is smaller, some of the destination groups can have multiple corresponding lanes. For instance, a two-lane road approaching a 4-way intersection can have one lane corresponding to vehicles destined to turn left on the intersection and the other lane corresponding to both right-turning and straight going vehicles.

Lane change problem can be derived from the lane sorting problem by considering the case where only one vehicle out of the group is attempting to change lanes and other vehicles are already in their destination lanes. Thus solutions to lane sorting problem can also be used for the purpose of lane changing of individual vehicles.

We next present the Cooperative Lane Sorting (CLS) algorithm for lane sorting of autonomous vehicles. First, we discuss the architecture used for this purpose.

4.2 Scenario Description

Consider a long stretch of straight, multi-lane road. Each lane in this road is drivable and is of the same width (not necessary). Long here means that we are not considering the distance deadline for now rather, we are considering that we have a road with some fixed number of lanes for infinite (sufficiently long) distance. The length of the road required for any particular traffic setting can then be determined from the results obtained. The road does have a start line. Incoming vehicles enter this road from the start line with some random velocity bounded by a speed limit of V_{max}. All vehicles are required to have SAE (Society of Automotive Engineers) level-2 autonomy [116] or more (as the vehicle will need to perform autonomous longitudinal and latitudinal movements to change lanes) and are capable of communicating with the Scenario Controller (SC) using wireless communication either directly or via Road Side Units (RSU). Scenario Controller is the central controller which contains the presented algorithm and is responsible for performing all associated tasks such as communication and computation.

As shown in Figure 4.2, the start line is followed by a road section called Velocity Transition Section (VTS). Incoming vehicles adjust their velocity in this section to achieve the common velocity (V_{common}). The length of VTS can be determined using Newton's equations of motion and the limits over velocity and acceleration of vehicles in the scenario. For example, for a maximum velocity of $60kmph$ and minimum acceleration (or deceleration) of $3m/s^2$, the required length of VTS will be 46.3 meters.

V_{common} is one of the two parameters that can be varied to suit the incoming traffic density and available road length. The other one is frame length (discussed later). Their values are taken from a lookup table that has previously been generated using the experiments shown in this chapter. In case when a sufficiently long stretch of road is available, V_{common} can be set to the V_{max}. Otherwise, V_{common} shall be decreased with a decrease in the length of the available road because reduced velocity will give more time to vehicles for adjusting their position.

By considering such a scenario, we demonstrate a possible architecture that can be realized before any scenario that is benefited by sorted traffic, for example, a Tollgate or an intersection, etc. In the experimentation, for demonstration purpose, we have performed simulations for V_{common} equals to 5m/s, 10m/s and 15m/s. These velocities in kilometers per hour correspond to 18, 36, and 54 kmph, which are very common for any urban traffic scenario.

All incoming vehicles perform wireless communication with SC and pass their information such as their length, width, velocity, route, current (incoming) lane (l_c), etc. For simplicity, we have assumed the length of every vehicle to be the same. SC in return passes V_{common} and the lane in which vehicle has to shift (destination lane, l_d). The destination lane of a vehicle may or may not be the same as its initial lane. Deciding the destination lane of incoming vehicles is a research topic in itself known in the research community by the name Lane Assignment. Lane assignment can have different approaches such as grouping by destination, dynamic grouping, grouping by size [18] etc. Since in this work, the grouping of vehicles is done based on the destination direction of each vehicle at the intersection, we can say that SC is using the grouping by destination strategy of lane assignment in the background. Although the considered scenario is an example of grouping by destination, the presented algorithm may well be used along with any other lane assignment technique. Vehicles start transiting to V_{common} as soon as they enter VTS. Lane sorting starts after vehicles have left

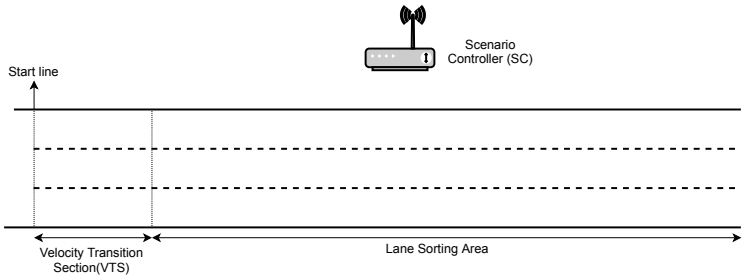


Figure 4.2: VTS and LSA in the Scenario

VTS and are now in the Lane Sorting Area (LSA).

Although the presented work is generalized in sense of the number of lanes i.e. there can be any number of lanes on the road, we have used for explanation purposes, a scenario with a road that has three lanes wherever appropriate. We have abstracted away from the imperfections in communication to keep the objective of this work clear and that is to propose a lane sorting algorithm hence the communication is assumed to be flawless. The other assumption of the presented work is that all vehicles in the scenario are capable of having autonomous control over their lateral as well as longitudinal movement i.e. the all the vehicles in the scenario have the autonomy of SAE level-2 or more. Also, for the sake of simplicity, we have kept the dimensions of all the vehicles to be the same (length = 3 meters and width = 2 meters).

4.3 CLS Algorithm

The sorting area and vehicles inside it are divided into various sections known as Frames. All frames are of the same size when created and move with a velocity the same as that of the vehicles contained in it i.e. V_{common}. As a result, the frame-vehicle association is always preserved unless the frame is to be merged with some other frame. Frames are created at the start of the sorting area. A frame after being created can merge with another frame if required and frames are destroyed when all their contained vehicles get past the lane sort area. In between their creation and destruction, they move with V_{common} velocity. All these frames are non overlapping and every vehicle in the scenario should be associated with a frame. Frames can have a gap in between them i.e. they are not necessarily required to be back-to-back as shown in Figure 4.3.

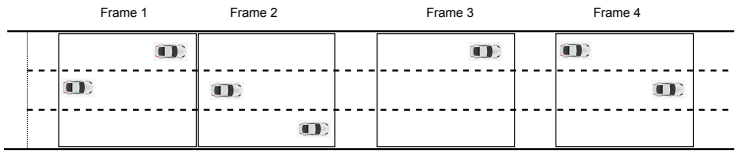


Figure 4.3: A possible distribution of frames

The geometrical center of the vehicle is taken as the position of a vehicle. As stated earlier, all vehicles and frames move with the same longitudinal velocity, when seen from a perspective of the moving frame i.e. when we see a frame from a reference moving along with the frame, then it will look stationary with fixed relative positioning of vehicles inside it. From this perspective, a lane change maneuver will look like the vehicle is moving horizontally in between the lanes given that the vehicle maintains this longitudinal velocity while changing the lane as well.

In continuation of the above observation, we can say that a vehicle can perform a safe lane change maneuver if it can obtain a horizontal space that is non overlapping with other vehicles. We call this horizontal space as $Channel$.

This reduces our task at hand at this stage to get independent channels for vehicles wanting to change lanes to their destination lanes. We make use of Linear Programming over the positioning of vehicles in the frame to achieve this task. But first, we discuss how frames are created in the scenario.

4.3.1 Frame Creation

To explain how frames are created, let us consider that initially there are no vehicles in the scenario. As vehicles start coming in the scenario, they are first added to a temporary list. A check is made at every step to get the position of the first vehicle in this temporary list. As long as this position is less than the frame length, all incoming vehicles are added to the temporary list as shown in Figure 4.4. The moment at which the first vehicle in the temporary list crosses the frame length distance, a frame is created and all vehicles present in the temporary list are assigned to the newly created frame. The temporary list is now cleared (Figure 4.5). This procedure is repeated when the next vehicle enters the scenario.

Once a frame is created, it starts moving with the V_{common} velocity. As a result, the relative

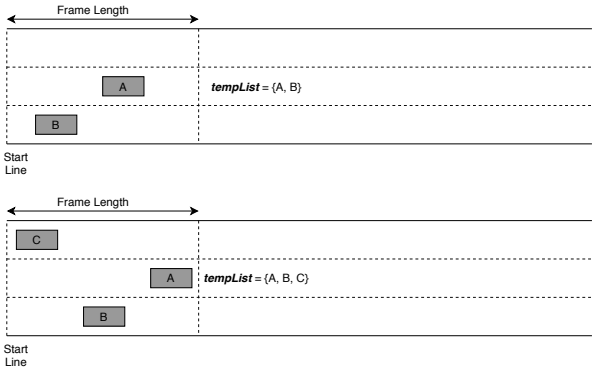


Figure 4.4: Frame creation

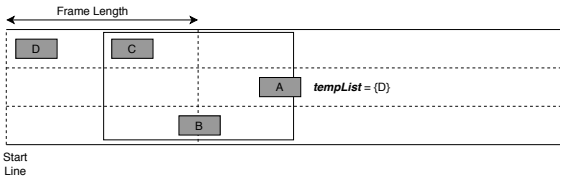


Figure 4.5: Frame creation

positioning of vehicles inside the frame shall be fixed unless vehicles are in a maneuver to change lane or getting into the assigned position in the frame. After vehicles complete the maneuver, they will again attain the V_{common} velocity.

As vehicles don't cross their frame and frames don't overlap, we can say that through frames, the complete scenario has been divided into various independent sections that can be processed in parallel. Processing each frame involves formulating a Linear Programming (LP) problem that will return vehicle positions as a result. In case the solution of the linear programming formulation is not feasible, the frame is merged with another frame. We will next discuss the LP formulation.

4.3.2 Problem Formulation

As discussed earlier, we breakdown the task of sorting vehicles on the entire road into sorting vehicles in smaller road sections called frames. We will now discuss the formulation of the linear programming problem which gives channel positions as its solution. We first define some sets of vehicles present in the frame under consideration. The first set is named *Vehicles*, which is the set of all the vehicles inside the frame. There are three subsets of the *Vehicles* set. These subsets are *SortedVehicles* set, *UnsortedVehicles* set and *SupportingVe-*

hicles set.

Let us first give the formal definition of the above sets and subsets.

Let the *Vehicles* set be defined as the set of all the vehicles inside the frame. If there are *m* vehicles inside the frame then,

$$Vehicles \;\; = \;\; (V_1, V_2, ..., V_m) \tag{4.3}$$

Next, the *SortedVehicles* set is defined as:

$$SortedVehicles \subseteq Vehicles \;\; \textbf{s.t.} \;\; \forall V_i \in SortedVehicles, CL(V_i) = DL(V_i) \tag{4.4}$$

Next, *UnsortedVehicles* and *SupportingVehicles* are defined as:

$$\begin{aligned} UnsortedVehicles \subseteq Vehicles \;\; \textbf{s.t.} \;\; \forall V_i \in UnsortedVehicles, \\ CL(V_i) \neq DL(V_i) \quad \textbf{and} \quad coop(V_i) = 0 \end{aligned} \tag{4.5}$$

$$\begin{aligned} SupportingVehicles \subseteq Vehicles \;\; \textbf{s.t.} \;\; \forall V_i \in SupportingVehicles, \\ CL(V_i) \neq DL(V_i) \quad \textbf{and} \quad coop(V_i) = 1 \end{aligned} \tag{4.6}$$

Here the *coop* relation gives the truth value for cooperation from the vehicle. If it is 0, vehicle will demand for a channel and if it is 1, vehicle will cooperate by not demanding a channel.

As their names suggest, *SortedVehicles* set contains those vehicles that are already in their destination lane, *UnsortedVehicles* set contains vehicles that are not in their destination lane and will demand a channel for lane-change maneuver, and *SupportingVehicle* set contains those vehicles which are not yet in their destination lane but are behaving as sorted vehicles temporarily. Unsorted vehicles always demand a channel whereas sorted and supporting vehicles always clear the space for creating channels. We will discuss later in this section how and when an unsorted vehicle is made supportive.

A Channel, as discussed earlier is the unoccupied space reserved for a vehicle $V_i \in$

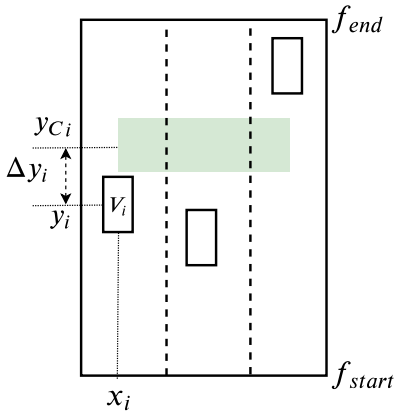


Figure 4.6: This figure shows a frame along with notations used in equations

$UnsortedVehicle$, spanning from $CL(V_i)$ to $DL(V_i)$. It is formally defined using the position inside the frame i.e.

$$Channel(V_i) \quad = \quad y \tag{4.7}$$

Where *y* is the position of the channel with respect to the frame start line, f_{start}.

Let us consider the frame shown in Figure 4.6. The frame starts at f_{start} and end at f_{end}. Let the vehicle i with initial position (x_i, y_i) be assigned a channel centered at y_{C_i}. Vehicle *i* will need to shift by a distance of Δy_i = ($y_i - y_{Ci}$) to get into its corresponding channel. To keep this shift minimum, we would want that the channel should be as close as possible to the vehicles' initial position. Along with this, we would also like to minimize the shift required in the position of the sorted and unsorted vehicles as well. This requirement gives us the following objective to our linear programming problem.

$$\textbf{Objective} : Min. \ \Sigma|\Delta y_i| \ \ \forall i \in Vehicles \tag{4.8}$$

We will now discuss associated constraints:

1. **Vehicles are always separated longitudinally by at least a safe distance.** We call this safe distance as Safety Gap (SG). This means channels, sorted vehicles and supporting vehicles should always be positioned such that vehicles will always have a

longitudinal physical separation of at least SG in between them. We have considered the value of SG to be 2 meters. The associated constraint will look like follows.

$$\begin{gathered} |(y_i + \Delta y_i) - (y_j + \Delta y_j)| \geq v\_len + SG \\ \forall i, j \ \in \textit{Vehicles},\ i \neq j \\ |l_{cj} \in (l_{ci}, ..., l_{di}) \ or \ l_{dj} \in (l_{ci}, ..., l_{di}) \end{gathered} \tag{4.9}$$

Here l_c and l_d represent the current lane and the destination lane respectively. The condition $l_{cj} \in (l_{ci}, ..., l_{di})$ *or* $l_{dj} \in (l_{ci}, ..., l_{di})$ select vehicles that can have conflict while changing lanes. For vehicles that can not possibly conflict with each other while changing lanes, this constraint is not applicable. v_len represents length of the vehicle.

2. **Vehicles associated with a frame should lie completely inside the frame.** This constraint makes all the frames independent because no vehicle is allowed to cross the frame boundary. The associated constraints will look like.

$$y_i + \Delta y_i \geq f_{start} + \frac{v\_len}{2} + \frac{SG}{2} \tag{4.10}$$

$$y_i + \Delta y_i \leq f_{end} - \frac{v\_len}{2} - \frac{SG}{2} \tag{4.11}$$

$\frac{SG}{2}$ is added in both constraints to prevent double safety spacing between vehicles near the boundary of two frames.

3. **Actual order of vehicles in lanes should be preserved.** This constraint keeps vehicles in the same lane in their actual ordering. Since the linear programming solution will

contain a change in position in the geometrical center of the vehicle, this constraint will put limits on the values of Δy such that actual order of vehicles is maintained.

$$\begin{aligned}(y_i + \Delta y_i) - (y_j + \Delta y_j) >= v\_len + SG \\ \forall i, j \in Vehicles \mid i \neq j, x_i = x_j, y_i > y_j\end{aligned} \tag{4.12}$$

$$\begin{aligned}(y_j + \Delta y_j) - (y_i + \Delta y_i) >= v\_len + SG \\ \forall i, j \in Vehicles \mid i \neq j, x_i = x_j, y_j > y_i\end{aligned} \tag{4.13}$$

As we can see the objective and constraints given in equations 4.8 and 4.9 respectively are not in a form appropriate for linear programming formulation because they contain absolute value functions. To resolve this we can either use a quadratic programming solver or we could transform these equations into a form acceptable by linear programming solvers. We choose the second option as using a quadratic programming solver is more resource-intensive than a linear one.

4.3.3 Transforming to Linear Programming

To transform these expressions into linear programming solver acceptable form, we need to add additional variables and expressions. We do this one by one starting with the objective itself. To transform the objective, we add an extra variable named $\overline{Obj}$ and a variable array $\overline{\Delta y}$ in which all the elements have a lower bound of zero, along with following additional constraints.

$$\overline{\Delta y_i} >= \Delta y_i \;\; \forall i \in \textit{Vehicles} \tag{4.14}$$

$$\overline{\Delta y_i} >= -\Delta y_i \;\; \forall i \in \textit{Vehicles} \tag{4.15}$$

$$\Sigma\overline{\Delta y_i} <= \overline{Obj} \quad \forall i \in \textit{UnsortedVehicles} \tag{4.16}$$

$$\Sigma\overline{-\Delta y_i} <= \overline{Obj} \quad \forall i \in \textit{UnsortedVehicles} \tag{4.17}$$

And our new objective will become

$$\textbf{\textit{Min.}} \quad \overline{Obj} \tag{4.18}$$

To transform the constraint given in equation 4.9 into an appropriate form, we need to add one binary variable (b) for each combination of i and j and a variable with a constant value (M) large enough to satisfy constraints. Every instance of equation 4.9 will be replaced by following two constraints.

$$(y_i + \Delta y_i) - (y_j + \Delta y_j) + M * b_{ij} \geq v\_len + SG \tag{4.19}$$

$$(y_j + \Delta y_j) - (y_i + \Delta y_i) + M * (1 - b_{ij}) \geq v\_len + SG \tag{4.20}$$

The value of M should be large enough to satisfy the above equations. We take it to be double the frame length, any larger value will also work fine. Now, this linear programming is solved for a solution. In the first iteration, all the unsorted vehicles inside the frame demand for a channel. If it is feasible for every unsorted vehicle to get a channel in the first iteration itself, the solution is obtained. However, if it is not possible for every vehicle to get a channel, the LP solver will report no feasible solution. This is where we are required to shift vehicles from the *UnsortedVehicles* list to the *SupportingVehicles* list. This could be

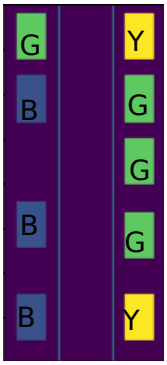


Figure 4.7: A case where frame merge is required.

done iteratively by shifting one vehicle to the *SupportingVehicles* list at each iteration either until all vehicles from the *UnsortedVehicles* set are made supportive or untill a solution is obtained.

If a solution is obtained, vehicles are required to move into their assigned positions. Unsorted vehicles are required to move into their assigned channels; whereas, sorted and supporting vehicles to the positions obtained by solving LP. Every LP solution is followed by a movement step in which all vehicles shift to assigned positions and then lane change maneuvers are performed. After the movement step is over, the next sorting step is started. This sequence of sorting and moving is repeated until all vehicles in the frame are sorted. When traffic is high or in the case when the number of vehicles destined to one lane is more than the lane capacity, it is not possible to get all vehicles in the frame into their respective lanes. In such a case, the frame is merged with the frame upstream. We can see an example in Figure 4.7. In this figure, the vehicles are shown as the smallest rectangle that will fit the vehicle when seen from above. Color of the rectangle means following:

- **Blue**: Vehicles destined to go to the left-most lane;
- **Cyan**: Vehicles destined to go to the middle lane;
- **Green**: Vehicles destined to go to the rightmost lane;
- **Yellow**: Vehicles already in their destined lane.

Figure 4.7 shows an intermediate stage while sorting a frame. At the stage shown, the three blue vehicles and three out of four green vehicles have reached their destination lane. As can be seen, the rightmost lane is now completely occupied and does not have

space to accomodate the remaining green vehicle. The solution of LP formulation done at this stage will thus be infeasible and as per the procedure presented above, the last green vehicle, which is also the only unsorted vehicle at this stage would be made supportive. The observation to be made from this example is that, whenever there is such a case where the number of unsorted vehicles is zero and the number of supporting vehicles is greater than zero, that would mean there is no feasible solution for the LP formulation and the frame has to be merged with other frame to proceed with sorting.

Thus to detect this case, we check for the number of elements in *UnsortedVehicles* set and *SupportingVehicles* set. If the number of elements in *UnsortedVehicles* set is zero and the number of elements in *SupportingVehicles* set is greater than zero, it would mean that the number of vehicles destined for a particular lane in the frame is greater that the lane capacity and thus the frame has to be merged with the adjacent frame. Since the frame upstream is more probable to be sorted, we merge any such frame with its upstream frame.

4.3.4 Choosing Supporting Vehicles

As mentioned earlier, when there is no feasible solution to the formulated LP problem, vehicles are shifted from *UnsortedVehicles* set to *SupportingVehicles* set. This shifting, however, is not done arbitrarily rather, we need to decide first what number of vehicles are to be shifted and in what order. If the number of vehicles to be shifted is not known beforehand, one vehicle would be shifted in every run followed by formulation and solving of the entire LP problem. And this will be repeated multiple times until the required number of vehicles is not shifted. Thus not knowing the required number of vehicles to be shifted would result in an unnecessary delay in computation. To prevent this, we first find out the required number of vehicles to be shifted into the *SupportingVehicles* set.

We now explain analytically the process of finding the required number of vehicles to be shifted into the *SupportingVehicles* set. Let the maximum number of vehicles that can be accommodated in one lane is $lane\_max$ and there is n number of lanes in the scenario. Now for each lane, we need to find the number of vehicles each lane has to provide occupancy for to perform lane change of vehicles. Each lane will have to provide occupancy for vehicles of three classes which are:

Table 4.1: Occupancy requirements from lanes in Figure 4.8

	Class 1 (Already present)	Class 2 (Destined)	Class 3 (Passing)	Total
Left lane	3	0	0	3
Middle lane	4	2	0	6
Right lane	1	1	0	2

1. Vehicles already present in that lane,

2. Vehicles in other lane and destined to that lane and

3. Vehicles in another lane that has to cross that lane to reach its' destination lane.

For each lane in the scenario, we then add the number of vehicle occupancies that have to be provided corresponding to these three classes; let this sum be called $num\_occupancy$. The number of vehicles that have to be shifted to the *SupportingVehicles* list (N) is then obtained by adding the number of vehicle occupancy each lane has to provide in excess to $lane\_max$ i.e.

$$\begin{gathered} N = \Sigma(num\_occupancy_i - lane\_max) \\ for\ i = 1 \ldots n\ ;\ \ num\_occupancy_i > lane\_max \end{gathered} \tag{4.21}$$

N is the number of vehicles that have to be made supportive to obtain a solution. In case this number is equal to the number of unsorted vehicles, at any stage, then there will be no further feasible solution and the frame will have to be merged to proceed with lane sorting. Table 4.1 gives the occupancy requirements from lanes in the example scenario shown in Figure 4.8. The number of vehicles to be shifted is the addition of difference with the maximum occupancy (which is 5 for the example case; with the length of each vehicle = 3 meters, SG = 2 meters, and frame length of 25 meters) for lanes having occupancy greater than maximum occupancy. Hence, we need to shift 1 vehicle to the *SupportingVehicles* set.

To get the number of vehicles to be shifted into the *SupportingVehicles* set, an efficient method will be to iterate through all the vehicles and increment the lane counter for each lane in between the current lane and the destination lane of that vehicle. Where lane counter

is an integer array with an equal number of elements as there are the number of lanes and initialized with zero in all the elements. This procedure is shown in Algorithm 5. This algorithm takes as input the array of vehicle objects of all the vehicles in the frame (*vehicle_list*), number of lanes in the scenario (*num_lanes*) and maximum occupancy of each lane in the frame (*max_occupancy*).

Algorithm 5: Algorithm to calculate N

```
Input: Vehicle Object array (vehicle_list), num_lanes, max_occupancy
Output: N
N = 0;
lane_counter[num_lanes] = {0};
i = 0;
while (i < len(vehicle_list)) do
    l_s = vehicle_list[i].current_lane;
    l_d = vehicle_list[i].destination_lane;
    if (l_s < l_d) then
        j = l_s;
        while (j <= l_d) do
            lane_counter[j] += 1;
            j += 1;
        end
    else if (l_s > l_d) then
        j = l_d;
        while (j <= l_s) do
            lane_counter[j] += 1;
            j += 1;
        end
    else
        lane_counter[l_s] += 1;
    end
    i = i + 1;
end
i = 0;
while (i < num_lanes) do
    if (lane_counter[i] > max_occupancy) then
        N += (lane_counter[i] - max_occupancy)
    end
    i += 1;
end
return N
```

Even after getting the number of vehicles that have to be made supportive, the decision

of choosing which vehicle to be shifted to the supporting vehicles list is also important. Choosing the wrong vehicles can result in a greater number of runs for sorting vehicles or it can also result in no solution at all. For instance, consider the frame shown in Figure 4.8. As we can see in this figure, all three unsorted vehicles (one green and two cyan) can not move into their respective lanes simultaneously because the middle lane has space to accommodate only one more vehicle. Hence one vehicle has to be made supportive. Suppose that green vehicle is made supportive. Now, since the green vehicle is required to behave as a sorted vehicle temporarily, it will not shift lane but will make space for one of the cyan vehicles and that cyan vehicle will then move into the middle lane. Now, in the next sorting step, the green vehicle will again be eligible for attempting to change the lane along with the leftover cyan vehicle. Now suppose the green vehicle is again made supportive, then, in that case, there will be no space left in the center lane to accommodate the leftover cyan vehicle. This will end up being a no solution case. On the other hand, if the leftover cyan vehicle is chosen as a supportive vehicle instead of green, the green vehicle would have got a channel to shift to the right lane, and then finally in the next sort step, the remaining cyan vehicle will get chance to change lane. This example shows that the selection of vehicles to be made supportive is also crucial.

To find which vehicles are appropriate to be made supportive we extend the method used to calculate the number of vehicles to be made supportive. We discussed that there are three classes of occupancy that a lane has to provide to allow lane changes to take place. Vehicles of the first class are already present in the lane and if any of those vehicles are made supportive then it will not reduce space requirements from the lane. On the other hand, if any of the vehicles from the second or third class are made supportive then that vehicle will not demand channel and hence will reduce space requirements from that lane. This is the principle behind selecting supporting vehicles.

For each lane, a list of candidate vehicles is generated by combining vehicles of second and third class as discussed above. Then from this list of candidate vehicles, as many numbers of vehicles are made supportive as there is the requirement of space in excess of the maximum occupancy limit from that lane. Let us take the example given in Figure 4.8. Since only the middle lane has an occupancy requirement in excess of maximum lane occupancy of 5, we create a candidate vehicle list for only that lane. This candidate list will contain the two cyan

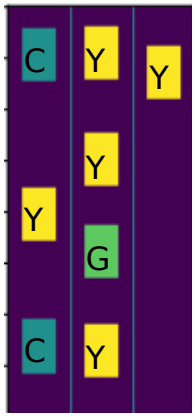


Figure 4.8: Example scenario to find number and order of vehicles shifted into *SupportingVehicles* set

vehicles out of which anyone can be chosen as the supportive vehicle.

Although the choice of supportive vehicles from the candidate vehicles can be arbitrary, an attempt should be made to select that vehicle first which is present in multiple candidate lists. Such vehicles will reduce the number of vehicles to be made supportive as they reduce the occupancy requirement of multiple lanes.

When all the discussed steps are followed, we will know beforehand whether the solution of the MILP formulation will be feasible or not. This will prevent unnecessary runs of the solver and thus will save time. When a solution does exist and is returned by the solver, then vehicles are required to move to the assigned positions. Supporting and Sorted vehicles will move to clear channels and Unsorted vehicles will move into their respective channel and later perform the lane change maneuver.

These steps are repeated iteratively until all the vehicles are sorted in the frame. At the start of each iteration, *Vehicles* set is divided into the three sub-sets, this means, supporting vehicles in one iteration will take part as unsorted vehicles in the next iteration initially and later the division will be done based on the requirements. Algorithm 6 depicts the complete workflow that goes into sorting one frame. This algorithm takes the frame object as input and makes use of the predefined operations as statements. For instance, in line 3, *SortedVehicles* set is obtained using equation 4.4, in line 5, N is obtained using Algorithm 5 and so on.

Algorithm 6: Sorting vehicles in a frame

Input: Frame object
Output: Sorted Frame

```
all_vehicles_sorted = False;
while !all_vehicles_sorted do
    Get SortedVehicles set;
    Get UnsortedVehicles set;
    Find N;
    if N < len(UnsortedVehicles) then
        Select and shift N vehicles from UnsortedVehicles to SupportingVehicles set;
        Solve optimization (LP) problem;
        if N == 0 then
            if new_frame == True then
                Rearrange vehicles in frame;
            end
            all_vehicles_sorted = True
        end
        Move vehicles;
    end
    else
        if new_frame == True then
            Rearrange vehicles in frame;
        end
        Merge frames;
    end
end
```

4.3.5 Frame Merge

As we have discussed above, when the distribution of vehicles in the frame is such that either there are a larger number of vehicles to be shifted into a lane than it's occupancy limit or all the unsorted vehicles are required to be made supportive, then the solution of the LP problem is not feasible. In such a case, the frame is to be merged with another frame. In our implementation, we have merged such frames with the frame upstream to that frame. This is because the frame upstream has spent greater time in the scenario and is more likely to be sorted. We have also put the restriction that the frame upstream should be sorted before it merges. The current frame will wait for the upstream frame to be sorted before it can merge and in the mean-time will just continue traveling with V_{common} velocity. When two frames merge, the f_{start} of the following frame is the start position of the new frame and the f_{end} of

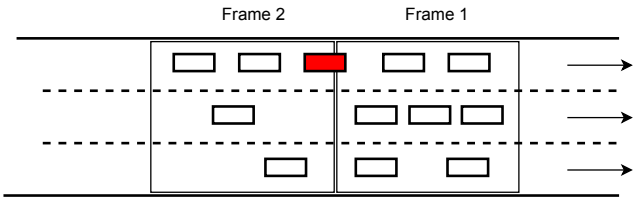


Figure 4.9: The figure shows a case where the rearrangement of vehicles will be required to prevent a collision. The red vehicle is associated with Frame 1 but protrudes into Frame 2 as it is not moved by the lane sorting LP formulation

the leading frame is the end position of the new frame. Since a sorted frame might help an unsorted frame in getting sorted, frames are never destroyed while they are inside the lane sort area.

In case the first frame in the scenario can't find a solution, then it will just increase its frame length to 1.5 times its current length.

4.3.6 Rearrange Vehicles in Frame

As can be seen in the Algorithm 6, when N is greater than or equal to the number of unsorted vehicles in the frame, we will be doing a frame merge operation. This is because the LP solver will not be able to give any solution for the positions of the vehicles in the frame and also until the frame ahead is not sorted, merge operation will be in a pending state. Now suppose, the frame that needs to merge has just been created and the last vehicle's front center is just inside the frame and the rest of the body of the vehicle is outside the frame as shown in the Figure 4.9. The last vehicle is in a dangerous position and will remain in that position because it's repositioning is only governed by the solution of the LP solver. Now since the solution of the LP solver is not feasible, all the vehicles in this frame will be traveling as they are in the frame. This creates a vulnerable scenario in which if a new frame is created just after this frame then the last vehicle will be inside the following frame's boundary and can cause a collision.

To prevent such a scenario, we have a routine that rearranges vehicles in a frame when the frame is created and it's N is greater than or equal to the number of unsorted vehicles in it or when all the vehicles in the frame are already sorted. The rearrange routine is placed as shown in Algorithm 6 (lines 11 and 19). It contains another LP formulation that only forces

vehicles to be inside the frame. The constraints are same as constraints given in Equations 4.10 and 4.11. And the objective is to minimize the overall vehicle movement. The variable *new_frame* is initialized with True when a frame is created and is set to False after it is processed by either the sort routine or the rearrange routine for the first time. Now a question might arise that what if even rearranging of vehicles inside a frame is not possible? This will only happen when the incoming vehicles are very tightly packed and the separation between vehicles is less than the safety gap (SG) that have been considered in the LP formulation. This puts a limit on the value of the safety gap we have used in our work. We can thus make the following statement:

The value of the safety gap (SG) that can be used in the implementation is limited by the spacing of vehicles in the incoming traffic. Or we can also say that the maximum value of the SG that we can use during lane sorting is the average spacing of vehicles in the incoming traffic.

Please note that the average spacing of vehicles in every frame in the incoming traffic should also be greater than or equal to the SG considered.

4.3.7 Shifting Vehicles Inside Frame

After a vehicle has been assigned a channel, it is required to move to that channel by making smooth changes in its velocity. Also, when it reaches the channel it should again have the V_{common} of the frame as its velocity. To perform this smooth change in position and velocity of vehicles, we make use of a proportional controller relation for updating velocity of any vehicle wanting to shift its position. Let us consider the scenario shown in Figure 4.6. The velocity while changing its position at each step of the simulation is given by following relations.

$$distance\_offset = y_{Ci} - y_i \tag{4.22}$$

$$velocity\_offset = V_{common} - vel \tag{4.23}$$

$$vel = vel + K_d * distance\_offset + K_v * vel\_offset \tag{4.24}$$

K_d and K_v are determined experimentally to result in smooth position shifts. The velocity of the vehicle is updated at every step using the relationship shown above. Please note that initially, *vel*, which is the instantaneous velocity of the vehicle, will be equal to V_{common}. Due to the *distance_offset*, the value of *vel* will change, however, the change in velocity will then be reverted to again attain V_{common} by the *velocity_offset* factor.

4.3.8 Adjusting V_{common} and Frame Length

We saw that when the safety gap is greater than the average spacing of vehicles in a frame, the solution to the MILP problem will not exist. This puts a limit on the value of the safety gap. Though we have considered a fixed safety-gap of 2 meters for simulations, it is not necessary to set it to a fixed value. Alternatively, it can be varied based on the traffic density and average spacing between vehicles. Also, since the vehicles participating in the cooperative lane sorting procedure are under tight coordination with each other due to minimal relative velocities, we can have lower values of safety-gaps. Such tight coordination of vehicles is also seen in platooning procedures and, there as well minimal safety gaps are required. Using a smaller value of SG will make the CLS algorithm applicable to higher traffic densities. For this reason, SG should only be large enough to satisfy the safety requirements.

To accommodate varying traffic, V_{common} is required to be adjusted accordingly such that the output rate of vehicles is always equal to the input rate. However, we need to come up with a mechanism that does not result in any disturbance in the CLS procedure. To devise such a mechanism, we only need to make sure that the relative positions of two frames should never decrease during or after velocity change. Now, there are only two possibilities for V_{common}. It can either be decreased or increased. Suppose, there are *n* frames in the scenario numbered *1, 2, ..., n* which are traveling with the velocity V_{common}. Now let us consider that new frames will now be assigned a velocity equal to $V_{common}\prime$ such that $V_{common}\prime < V_{common}$. That means the frame *n+1, n+2, ...* will have a velocity of $V_{common}\prime$. Now the two boundary frames corresponding to two different V_{common}s will have

an ever-increasing relative separation. So, decreasing V_{common} at any time is safe.

On the other hand, increasing V_{common} will result in a decreasing separation in frames. For that reason, we will have to make every frame *(1, 2, 3, ..., n, n+1, ...)* travel with $V_{common}\prime$. Frames with frame number *1, 2, ... n* that were earlier traveling with V_{common} will now transit to the velocity $V_{common}\prime$. Frames that are undergoing sorting will continue to do so but with the new common velocity. As we have shown, the shifting mechanism of vehicles does not depend on common velocity rather on the relative positions and velocities, change in common velocity will not affect the lane sorting procedure.
To vary the output rate of vehicles, other than varying the common velocity, we can also vary SG within the allowed range.

4.3.9 Complexity Analysis of CLS

The CLS algorithm has been shown to be frame-centric. Thus, the complexity of the overall algorithm is proportional to the complexity of the procedure to sort a single frame. Let us therefore analyse the complexity of one frame sort procedure. We will be using Algorithm 6 by considering one step at a time and getting its complexity.
Algorithm 6 starts with setting a boolean variable named *all_vehicles_sorted* to False. Untill it is true, the while loop on line number 2 is repeated. Which means the complexity of Algorithm 6 is *K* (a finite number) times the complexity of one iteration of the while loop.
Moving ahead, line number 3 and 4 give the set of sorted and unsorted vehicles. Both these are obtained by one traversal of the list of vehicles in the frame. Hence, the combined complexity of line number 3 and 4 is proportional to number of vehicles in the frame i.e. of the order of *O(m)*; where *m* is the number of vehicles in the frame.
Line 4 gives *N*, the number of vehicles to be shifted from *UnsortedVehicles* to *SupportingVehicles* set. The algorithm to obtain *N* is given in Algorithm 5 and it involves traversing through all the vehicles in the frame once. Hence the complexity of line number 4 is of the order of *O(m)* as well.
Moving ahead, we can see that either the line number 6 to 16, corresponding to the condition in *if* statement or line number 17 to 22 corresponding to condition in *else* will be executed. Let us first discuss the complexity of *if* block.

The *if* block will execute if *N* is less than the number of elements in *UnsortedVehicles* set. Next, if *N* is nonzero, the LP problem is formulated and solved to obtain solution.

If *N* is nonzero, the LP problem is formulated and solved to obtain solution. Or else if *N* is 0 and frame is just created, the rearrange routing is invoked which also involves formulating an LP problem and solving it. And if *N* is 0 and frame is not a new frame, then it would mean that the frame is sorted.

This implies that the complexity of *if* block is proportional to the complexity of obtaining linear programming solution. As the complexity of mixed integer linear programing is in polynomial order with respect to the number of variables in the worst case [117], complexity of the *if* block is in polynomial order with respect to the number of variables or number of vehicles in the frame i.e. $O(m^k)$, where *k* is a constant.

Similarly, the complexity of the *else* block is also of the order of $O(m^k)$ as it also involves one LP formulation.

Adding all the individual complexities, the overall complexity of Algorithm 6 is given by:

$$O(K_1.(m + m + m^k))$$

In a long run, the complexity can be said to be of the order of:

$$O(m^k))$$

Since the while loop will only run for a finite number of times, the complexity of sorting one frame is of the order $O(m^k))$.

4.4 Simulation

The algorithm presented considers a general road with n number of lanes. Every vehicle has a current lane and a destination lane which may or may not be the same. Vehicles are spawned into the system on a random lane. However, in the simulation we consider a scenario consisting of three lanes. The road is assumed to be destined towards an intersection such that the vehicles in the scenario are destined to go either right, left, or straight. Thus the objective of the algorithm is to sort vehicles into the lane corresponding to the destination direction of vehicles i.e. vehicles destined to go right should be brought to the right lane, vehicles destined to go left should be brought to the left lane and the vehicles destined to go straight should be brought on the middle lane. Vehicles may arrive on any lane which may

or may not be the same as their destination lane.

The algorithm may well be applied to a scenario in which the number of destination direction is not equal to the number of lanes on the road. In such a case, one or more lanes will carry the traffic of vehicles corresponding to multiple destinations. For instance, a road with two lanes, destined towards an intersection, can have one exclusive lane for the left turn and a shared lane for right turning and straight going vehicles.

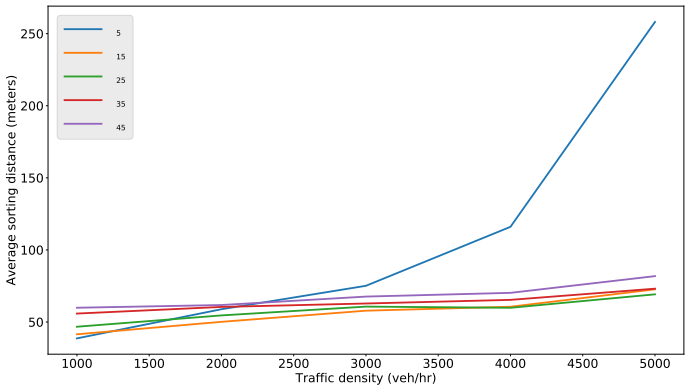
For simulation, we have used the Simulation for Urban MObility (SUMO) simulator which is an open-source, microscopic road traffic simulator. SUMO accepts the scenario, network, and route information in .xml format. The control logic is implemented inside a Python script which is interfaced with the SUMO simulator using the value retrieval and value setting functions given in the Traci library provided by SUMO. We have used the Mixed Integer Linear Programming (MIP) solver for formulating and solving linear programming problems. It is included in the Python package index and requires Python 3.5 or newer [118]. In the simulation, we have kept the length of the road to be sufficiently large and then recorded the average distance required by vehicles to get sorted. For simulations, we have generated 10 sets of random route files with 50 vehicles corresponding to traffic densities of 1000, 2000, 3000, 4000, and 5000 vehicles per hour. The frame length is varied in the range of 5 to 45 meters in steps of 10. Variation of frame length corresponds to varying occupancy limit from 1 vehicle to 9 vehicles in the frame as with the values of vehicle length (3 m) and safety gap (2 m) considered, one vehicle will need a space of 5 meters. The final results are generated by averaging the results of 10 sets of simulations.

The distance required to sort vehicles in lanes depends on three factors that are: (i) Inter-spacing of consecutive vehicles in the scenario, (ii) Traversal velocity of vehicles, and (iii) Frame size. Let us first discuss how these three factors affect the lane sorting distance. The CLS algorithm utilizes free space in a frame for moving around vehicles. For high traffic density in which vehicles are densely packed, there is less free space available. As we have seen when frames are densely packed, the number of iterations it takes for a frame to get sorted increases as the procedure is repeated until we have an empty *SupportingVehicles* set. Thus the time of sorting to sort a frame is higher in such a case. The second factor comes into the picture here. For a frame taking a longer time to get sorted will travel a greater distance if its velocity is large and vice-versa. Lastly, the effect of the frame size on sorting

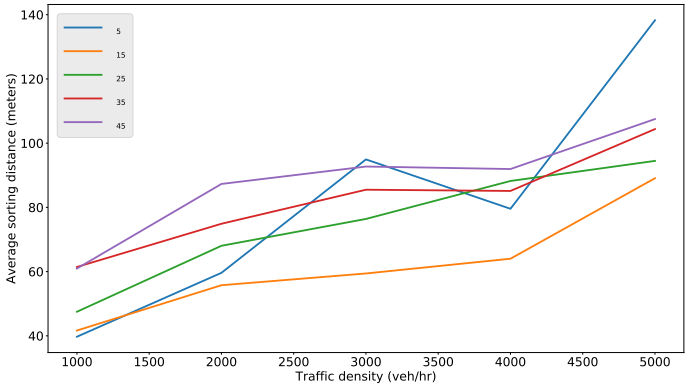
distance is due to the amount of space available for vehicles to move. Thus, for a densely packed frame, having a larger frame size is favorable. However, having a larger frame size with sparse traffic will result in greater sorting distance because vehicles will be required to move a distance equal to the frame length before they can start the sorting procedure.
Put in simple words, we require some extra space in between vehicles for performing the lane change operations. The greater this space in the incoming traffic, the smaller should be the lane sorting distance. In case when this space is not available in the incoming traffic or when the space present is not sufficient for efficient lane sorting, we have an option of increasing the frame size to capture more extra space. However, as discussed above, having too large a frame size can also increase the sorting distance.
Keeping the effects of the factors discussed above in mind let us discuss the results obtained.

Variation of the average distance to sort with respect to traffic density and frame length can be seen in Figure 4.10. In these figures, on the abscissa, we have varying traffic density and on the ordinate, we have average sorting distance. Each line in plots corresponds to different values of frame length.

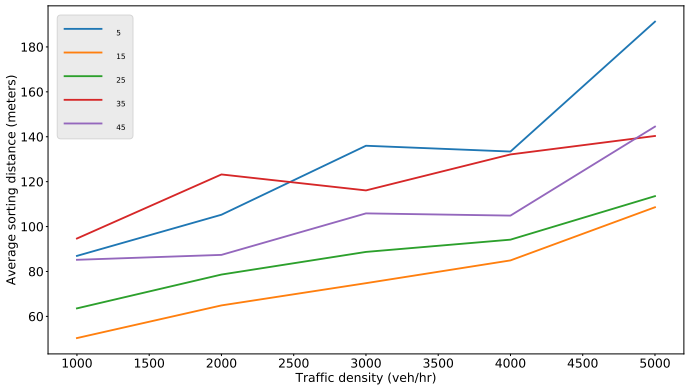
All these graphs show an increasing trend of average sorting distance with respect to traffic density for each frame length. However, the frame length of 5 meters has a profound behavior and this is because this length corresponds to the occupancy of 1 vehicle in each lane. For very low traffic in which vehicles are very sparse, this frame length is most suitable because vehicles will rarely have conflicts and also will be able to perform lane change as soon as they enter the scenario. Thus this frame length will result in minimum sorting distance. On the other hand, at very high traffic, a frame length of one will result in an excessive number of frames which in turn will result in an increased number of frame merge requests. Since frames can't be merged unless the frame upstream is not completely sorted, this will result in additional distance traveled while waiting for the upstream frame to get sorted. Thus frame length of 5 meters will result in maximum sorting distance at very high traffic.
As we discussed earlier that the length of the frame should only be sufficient to capture enough extra space and having a frame length larger than that will result in a greater value of sorting distance. Evidence of this can be found by looking at the sorting distance corresponding to different frame lengths for the traffic density of 5000 vehicles per hour in the three plots of Figure 4.10. In each of these plots, the minimum sorting distance is obtained



(a) Variation of average sorting distance with respect to varying traffic density for V_{common} = $5m/s$



(b) Variation of average sorting distance with respect to varying traffic density for V_{common} = $10m/s$



(c) Variation of average sorting distance with respect to varying traffic density for V_{common} = $15m/s$

Figure 4.10: Average sort time vs. traffic density for different values of V_{common}

for an intermediate value of frame length. Frame lengths smaller or greater than that will result in greater average sorting distance. Apart from the lowest and highest traffic densities, the same observations can also be made for intermediate traffic densities, for instance, the average sorting distance corresponding to the traffic density of 3000 vehicles per hour for different common velocities and frame lengths are shown in Figure 4.11 Following thumb-rule can be given from the above observations:

Appropriate frame length for any traffic density will be the one that will result in a good balance between the number of vehicles in the frame and free space present inside the frame which vehicles can use to get sorted.

For traffic density other than 1000 vehicles per hour, this balance is found at a frame length greater than 5 meters.

The most appropriate choice of V_{common} for any given traffic density and length of road available can be determined using the discussed experimentation. Using these experiments, we can also generate a look-up table that would give the most appropriate frame-length and common velocity for the given traffic density and available road length. We have considered common velocities of 5, 10, and 15m/s only for demonstration, however, the experimentation can be repeated for finer values of velocities and also for frame lengths. This look-up table can then be referred to in the future to decide the frame length and V_{common} that has to be kept so that vehicles incoming with the given traffic density sort themselves within the available road length. This look-up table based approach is very efficient as there is no need for additional processing in addition to traffic monitoring, hence less computation and infrastructure is required.

4.5 Discussion

The CLS algorithm is presented in this chapter to rearrange the incoming traffic such that every vehicle reaches its desired lane. The CLS algorithm is simulated for a range of traffic densities and varying adjustable parameters such as common velocity and frame length. The distance required in each simulation run is recorded in a look-up table that is referred to decide parameter values (*frameLen,* V_{common}*, and SG*) for the input traffic density such that the sorting distance is less than the available road length.

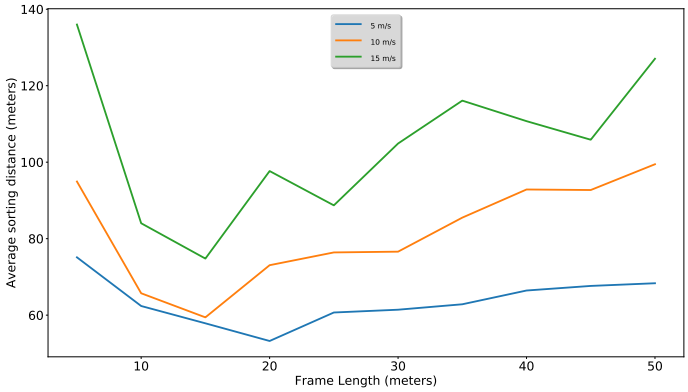


Figure 4.11: Variation of average sorting distance with respect to varying frame length for traffic density = 3000 Veh/hr

The CLS algorithm presented in this chapter is set-up according to the architecture considered. Since in the considered architecture, we have a straight road to which there is incoming unsorted traffic at one end and sorted traffic at the other end. To suit this setup, we defined the frame creation, frame merge, and frame rearrangement routines to cater to the needs of the architecture assumed. If we observe, we can say that sorting one frame is central to the CLS algorithm and is its basic building block.

This one frame sort routine can be used in some other architecture setup as well given that appropriate considerations are done for the safe and efficient implementation. The CLS algorithm being a generalized algorithm in terms of the number of vehicles wanting to change lanes, it offers us the freedom to apply it even for an individual vehicle lane change problem with coordination of other vehicles in its frame. Obviously, the rest of the vehicles will behave as supporting vehicles.

When we consider sorting one frame as the simplest unit, we can also redesign the communication and computation architecture to suit the requirements of the given scenario. For instance, the one frame sort routine can be used over an ad-hoc frame created by a speed synchronized group of vehicles traveling on a highway out of which some of the vehicles are wanting to change lanes with the cooperation of the rest of the vehicles in that frame. Most importantly, in any such setup, the ad-hoc frame creation should always satisfy the independence of the frame at all times.

In addition to the freedom, CLS offers in terms of redesigning the applicability, num-

ber of vehicles wanting to change lanes in a frame, and communication and computation architecture, the following advantages are associated with it:

- A minimum velocity change is exercised while changing lanes, other than that vehicles are required to maintain a fixed velocity. This ensures passenger comfort and since there are minimum acceleration and deceleration, there is minimum wastage of fuel and minimum exhausts from the vehicle.

- The frame-based architecture of CLS keeps batches of vehicles independent. This also makes their computation independent enabling parallel processing of individual frames. Such independence allows us to have flexibility in terms of computation architecture as well (centralized or decentralized).

- We have not specified how vehicles move laterally to change lanes. They are only required to maintain the longitudinal velocity to be constant. Thus vehicles have the freedom to have the desired steering control to change lanes as allowed by the horizontal jerk and passenger comfort requirements.

- Considering a fixed common velocity may feel like a harsh requirement but we believe it is very practical even for conventional vehicles with the present technology such as ACC and CACC. Having this requirement gives an additional advantage and that is the minimized relative motion of vehicles. This is advantageous as we can have lower safety gaps in between vehicles thus resulting in the packed arrangement of vehicles and as a result, improved capacity of roads.

- The application of MILP to solve for channel and vehicle positions results in very less computational requirements by the CLS algorithm. Added with the applied mechanism to accurately determine the number and choice of vehicles to be made supportive, keeps the required iterations to a minimum. This makes the proposed algorithm fit for real-time use.

- Since the frames are kept independent from each other by the design and by the frame rearrangement routine, delay in computation for one frame will never lead to collision threats.

4.6 Summary

In this work, we approached the unexplored problem of cooperative lane sorting among autonomous vehicles. The lane sorting problem is presented as a group task that all the vehicles in the scenario perform cooperatively. The presented algorithm first simplifies the problem by restricting each vehicle to maintain a fixed velocity that is common for all the vehicles in the scenario. This allows us to formulate a non-linear programming problem reducible to linear programming. Application of this linear programming formulation along with a well-structured skeleton of the algorithm that uses a frame-based approach on straight road results in a collision-free lane sorting of vehicles. The algorithm presented except assuming a straight road does not assume any particular scenario rather is generalized in terms of the number of lanes, incoming traffic density, and width of lanes. We do specify the safety-gap used in this work and discuss how it depends on the inter-spacing of vehicles in the incoming traffic. The implementation of the proposed algorithm is performed in the SUMO simulator and results are presented for different values of traffic density, velocity, and frame length. The experimentation presented can be used to decide the frame length and V_{common} for a given traffic density and available road length.

In the next chapter, we will present the HAIM algorithm for intersection management of AVs using a velocity based heuristic algorithm.

Chapter 5

Heuristic Autonomous Intersection Management

This chapter presents the proposed Heuristic Autonomous Intersection Management (HAIM) algorithm for intersection management of autonomous vehicles. The HAIM algorithm is a velocity-based algorithm that makes use of 4 levels of heuristics to resolve conflicts. We will next present the architecture considered, assumptions made, and behavior of vehicles and the intersection manager, and after that, we present the HAIM algorithm.

5.1 Architecture

5.1.1 Physical Architecture

Let us consider two straight roads carrying bi-directional traffic. Both the roads have three lanes dedicated to both directions of traffic. Now suppose these roads intersect, resulting in an intersection with 4 arms. These arms can be associated with directional specifiers such as East, West, North, and South. Each of these arms contains a total of 6 lanes with three lanes each for incoming and outgoing traffic to the intersection. Before the intersection, we have an approach length in all 4 directions which is divided into two parts. First, we have a buffer region and then we have an approach region with the approach region being closer to the intersection. The Buffer Region starts at the Buffer Start Line (BSL), and ends at the Approach Start Line (ASL) and the approach region starts at the ASL and ends at the

Intersection Start Line (ISL). On the outgoing side of the road, we have the Intersection Exit Line (IEL) as the boundary between the intersection region and the depart length. The depart length is also divided into two regions. The region closer to the intersection is called the Rapid Depart Region (RDR). RDR starts at the IEL and ends at the Virtual Depart Line (VDL) and after the VDL, we have the depart region. This architecture is shown in Figure 5.1, where we can also see the values of each of the lengths considered in this work.

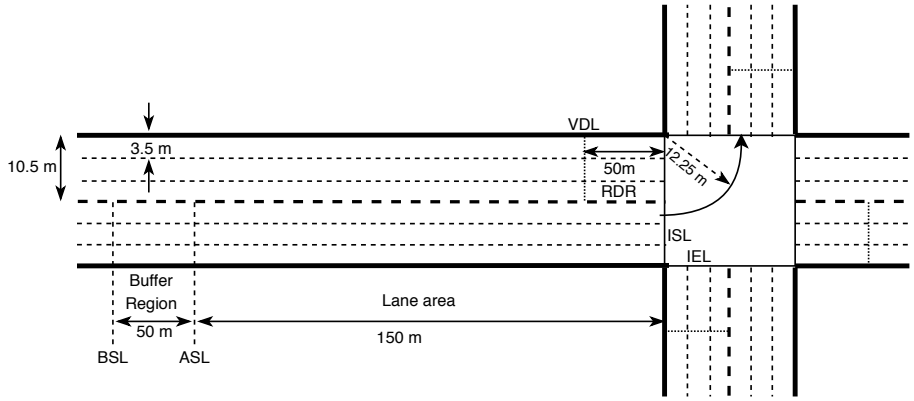


Figure 5.1: Architecture constants

We will use the term "Intersection Scenario" to denote the whole scenario consisting of the intersection region, approach regions, and the buffer region.

5.1.2 Communication Architecture

The intersection scenario is equipped with two types of communication agents. The first one is known as Road Side Units (RSU). RSU are present along buffer start line in all four directions. The other one is a central controller which we call the Intersection Manager (IM) and it is placed at the intersection region. There is a third player as well in the communication setup and that is the Connected and Autonomous Vehicle (CAV) itself. All three players of communication i.e. CAV, RSU, and IM perform wireless communication to exchange information. Vehicles and IM only communicate with the Road Side Units whereas Road Side Units communicate with both Vehicles as well as the IM. Thus RSUs are the mediators in the communication between vehicles and the IM.

Alternatively, we can also have an architecture in which no RSU is required. This is possible because the range of wireless communication in WAVE protocol can be a maximum of 1000 meters. In such architecture, vehicles will directly communicate with IM and vice-versa. Outgoing message from every vehicle contains information about the vehicle specifications (which includes the Vehicle's unique identifier), route plan, and arrival details (arrival velocity and arrival time). On the other hand, the outgoing message from the IM contains the vehicle identifier, specifying the vehicle for which the message is destined, and the target velocity for that vehicle. This target velocity is calculated by the IM using the Heuristic Autonomous Intersection Management (HAIM) algorithm presented in this chapter.
Before proceeding let us first get a rough estimate of the communication delays associated with the considered scenario.

Communication Delay Estimates:

Following are the distances that are used in the scenario.

1. Buffer Length (BL): 50 metres
2. Approach Length (AL) : 150 metres
3. Total Intersection Length (IL) : 21 metres

If IM is placed at the center of the intersection area than total distance between the vehicle at the start of the buffer region and the IM will be given by:

Distance (D) = BL + AL + IL/2 = 210.5 meters

For a round trip, the total distance will be double this figure (i.e. 421 meters). There can be some additional distance if the IM is placed at some height. Let us round this to 500 meters for the round trip distance.

For wireless communication:

It is possible to have wireless communication in vehicular applications for up to a distance of 1000 meters [119]. We could have an architecture in which all the vehicles will directly communicate with the IM.
Wireless communication takes place with the velocity of light which is given by $c = 3 \times 10^8$. The minimum communication delay will then be given by:

$$Delay_{wl} = \frac{500}{3} \times 10^{-8} = 1.66\ \mu sec$$

With RSU:

Since the considered scenario is using RSU for communication, a possible architecture would be that an RSU is connected near the start of the buffer region and which is communicating with the IM via a copper wire. In this case the majority of communication delay will be caused by the current speed in the wire.

The speed of current in wire can have a velocity in the range 50-99% of the velocity of light. Let us take the worst case of 50%. If the RSU is connected at the start of the RSU then the distance of the wire will be around the same as considered for wireless communication. Then the time taken by one round-trip of message will be double the time taken by wireless communication, i.e.,

$$Delay_{wired} = 3.33\ \mu sec$$

Now let us suppose that the RSU is connected at the start of the buffer region on the side of the road. For a scenario with three lanes and lane width of 3.5 meters, distance of a vehicle from the side of road will be less than 10.5 meters, let us assume it to be 10 meters. Also, the RSU can be placed at a height, let us assume the height to be maximum 10 meters. The distance between the vehicle and RSU will then be:

$$Dis_{R-V} = 10\sqrt{2} \approx 15 \text{ meters}$$

Round trip time delay in communication with RSU = 30/c = 0.1 μsec

Now let us suppose that the vehicle is traveling with its maximum allowed velocity of 17 metres/sec. Then the distances covered in these time intervals are:

- **No RSU**

  Communication is between vehicle and central controller:

  Distance = 17 m/s * 1.66 μ sec = 0.002822 cm.

- **With RSU**:

  - Communication between vehicle and RSU:

    Distance = 17 m/s * 0.1 μ sec = 0.00017 cm

  - Communication between RSU and central controller:

    Distance = 17 m/s * 3.33 μ sec = 0.005661 cm

- Total round trip delay via RSU on both the ways:

  Distance = 17 m/s * (3.33 + 0.1) μ sec = 0.005831 cm.

These delay estimates are calculated to make sure that the buffer length considered is sufficient for multiple communication hops in case of imperfect communication.

5.2 Assumptions

The following assumptions are made while proposing the HAIM algorithm.

- We assume traffic that only consists of CAVs. In the proposed HAIM algorithm, vehicles will require autonomous control over both longitudinal as well as latitudinal movement to traverse the fixed trajectory inside intersection thus requiring SAE Level-2 or higher autonomous vehicles.
- Every vehicle communicates with IM via RSUs to obtain the target velocity by the time vehicle is in the buffer region. In this way, we abstract away from the communication and computation related delays and assume that all communication and computation related tasks are completed while the vehicle is in the buffer region.
- The incoming traffic is sorted. By sorted we mean that vehicles are traveling on the lane corresponding to their destination direction i.e. vehicles destined to go left at the intersection travel on the left lane, vehicles destined to go straight at the intersection travel on the center lane, and vehicles destined to go right at the intersection travel on the right lane. This rules out the possibility to overtake in the approach region as well. U-turns are not allowed at the intersection.
- For the sake of simplicity, we have assumed all vehicles are of the same type i.e. they have the same length, breadth, and acceleration and deceleration rate.

Restrictions on the movement of vehicles in lane bring discipline in traffic and organize it. Since vehicles travel in a specified lane in the approach region and also the approach lane has a corresponding depart lane, we have a one to one correspondence between arrival and departure lanes. This one to one correspondence, along with fixed trajectories of

lane-to-lane connections will reduce the possible conflicting areas to the overlapping areas of these lane to lane connections. We would have conflict-free intersection management if we can schedule vehicles such that there is no overlap in the occupancy at any shared conflict point by two vehicles. The intersection control task is thus reduced to scheduling vehicles at these conflict points.

5.3 Vehicle Behavior

Consider a vehicle traveling safely before the buffer start line. It will cross BSL first to enter the buffer region. Let us say this happens at the moment *enterTime* and the vehicle has a velocity of *enterVel*. As soon as the vehicle enters the buffer region, it is required to send a message packet containing its identification, physical parameters which are its length and width, movement parameters which are acceleration and deceleration rate, its arrival parameters which are its arrival time and arrival velocity, and its route plan parameter which is its destination direction at the intersection. Since we have a one to one correspondence between the arrival lane with the depart lane, we can also specify the arrival lane in place of the route parameter. Hence the message packet the vehicle will send to the IM will contain the following information.

$$< vehicleId, length, width, arriveLane, arriveVel, arriveTime, acc, dec >$$

Please note that *arriveTime* and *arriveVel* are the time and velocity of the vehicle at the moment it is going to cross the approach start line (ASL). Thus *arriveTime* is not same as *enterTime* and *arriveVel* is not always same as *enterVel*. To prevent any collision inside the buffer region, velocity of vehicles in the buffer region is limited by the velocity of the vehicle present inside buffer region ahead in the same lane. If there is no vehicle ahead in the same lane inside the buffer, the *arriveVel* will be same as the *enterVel*. After we get the *arriveVel*, *arriveTime* is evaluated using the Newton's laws of motion. Please note that the information sent by a vehicle to the IM are saved by the IM and refered using the *vehicleId* when needed for instance *enterTime[vehicleId]* will refer to the enter time of the vehicle with *vehicleId* as

its id.

By the time vehicle is in the buffer region, it will receive the target velocity (V_{target}) from the IM. As soon as the vehicle crosses ASL, it will start transiting from the *arriveVel* to the V_{target}. When the vehicle attains the V_{target}, it will maintain the velocity throughout the rest of the approach region and the intersection region. After exiting the intersection region, vehicle is required to attain the maximum allowed velocity in the scenario (*maxVel*). As soon as vehicle crosses the IEL, it will accelerate to the *maxVel* and travel with it thereafter. Figure 5.2 shows a possible velocity profile of a vehicle in the intersection scheduled by the HAIM algorithm.

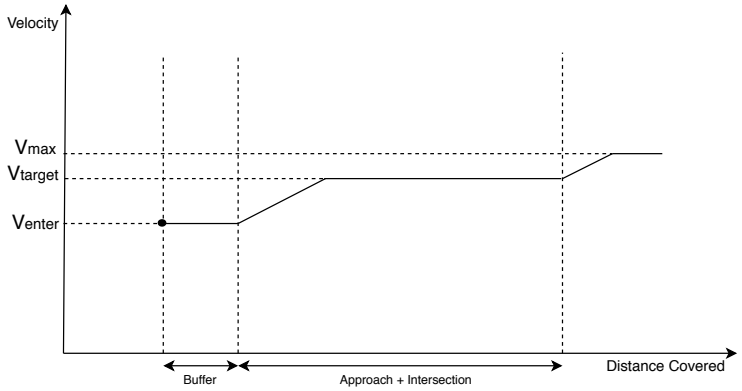


Figure 5.2: A possible velocity profile of vehicle

5.4 Intersection Manager Behavior

The job of the intersection manager (IM) is to calculate the V_{target} for each incoming vehicle using the HAIM algorithm. The pipeline for finding the V_{target} for any incoming vehicle starts when IM receives any packet from a new incoming vehicle. This velocity will be transmitted back to the vehicle once it has been calculated. Requests for calculating V_{target} are processed in the sequence of the arrival of the message packet from vehicles. Since we assume that there is no packet loss, the arrival order of packets is the same as the arrival order of vehicles inside the intersection scenario. Thus, the IM processes the velocity requests sequentially in the order of the arrival of message packets from vehicles.

To perform the above-mentioned task, the IM is required to have the exact values of the infrastructure parameters such as lane width, approach length, positions of the conflict points,

Table 5.1: Information available with the IM

	Variable Name Used	Detail
Vehicle Information	arriveVel	Velocity of vehicle at ASL
	arriveTime	Time while crossing ASL
	arriveLane	Lane id of the vehicle
	cp1, cp2, cp3, cp4	Indices of conflict points in vehicle path
Lane Information	lastVehicleDepartTime	Depart time of previous vehicle in the lane
	intersectionPathLen	Length of path in intersection for the lane
Conflict Point Information	cp_record	Reservation record at conflict point
Infrastructure Information	approachLen	Length of the approach
	departLen	Length of the RDR
	maxVel	Speed limit for the scenario

departure point, etc. Along with this, the IM stores information about the arrival details of every vehicle, reservation record for every conflict point and depart points, output velocity of each completed heuristic layer, etc. All details related to a particular vehicle can be accessed using its identification number as the index. Table 5.1 contains the information IM has at its disposal. Vehicle, lane and conflict point information are accessible using the index of the vehicle, lane or conflict point respectively. Infrastructure variables are present as global variables.

5.5 Motivations for the HAIM algorithm

The characteristic features of the HAIM algorithm based on the above description of the architecture under consideration and behaviors of vehicles and the IM are the following.

- HAIM is a heuristic algorithm. It makes use of four levels of heuristics which successively resolve all conflicts a vehicle can have in the approach region, intersection region, and depart region.
- The HAIM algorithm provides a velocity that has to be attained by the vehicle as soon as it enters the approach region and then maintained throughout the rest of the approach and the intersection region. This makes the HAIM algorithm a Velocity based approach.
- The intersection is modeled as a set of conflict points and the intersection management

is reduced to the scheduling of vehicles at these conflict points.

The motivation behind proposing a heuristic algorithm for intersection management comes from the fact that this is a real-time control problem that requires strict compliance of timing constraints. The computational requirements will increase with the increase in the number of vehicles in the scenario. An efficient and less computation-intensive algorithm will always be a better option for such scenarios. We have seen various algorithms in the literature survey that makes use of computationally intensive procedures such as optimization. Furthermore, only a few articles have attempted to meet the real-time requirements of this problem. The HAIM algorithm, being a heuristic one does not need any computationally intensive procedures and implements rule-based scheduling of vehicles.

In the HAIM algorithm, vehicles are required to maintain a constant velocity given by the IM after crossing the ASL. This also means that HAIM implements a stop free intersection traversal of vehicles preventing the formation of queues before the intersection. The constant velocity also reduces the energy requirements of the vehicles by preventing unnecessary acceleration and deceleration phases. As a result, this results in the reduction of the emission of harmful gases and improved economy of the vehicle.

Autonomous vehicles have far more accurate control of their movement with respect to that of a manually controlled vehicle. They are also obedient and will always respect the set of instructions they are given. This demonstration of discipline by the autonomous vehicles encourages us to trust their ability to traverse a given trajectory defining the lane to lane connection in the intersection region. Vehicles are already sorted according to their destination direction and added with the predefined trajectory, we will have a reduced version of the problem we are dealing with. When the lane discipline and trajectory conditions are placed, intersection reduces to a set of 16 conflict points with 4 conflict points in each lane to lane connection (except right-turning lanes, they have no conflict) as shown in Figure 5.3. We can also say that the entire intersection has been broken down into separate lane to lane intersections. This will greatly reduce the state space of the problem and thus will facilitate devising an efficient intersection control algorithm.

We will next describe the Heuristic Autonomous Intersection Management (HAIM) algorithm.

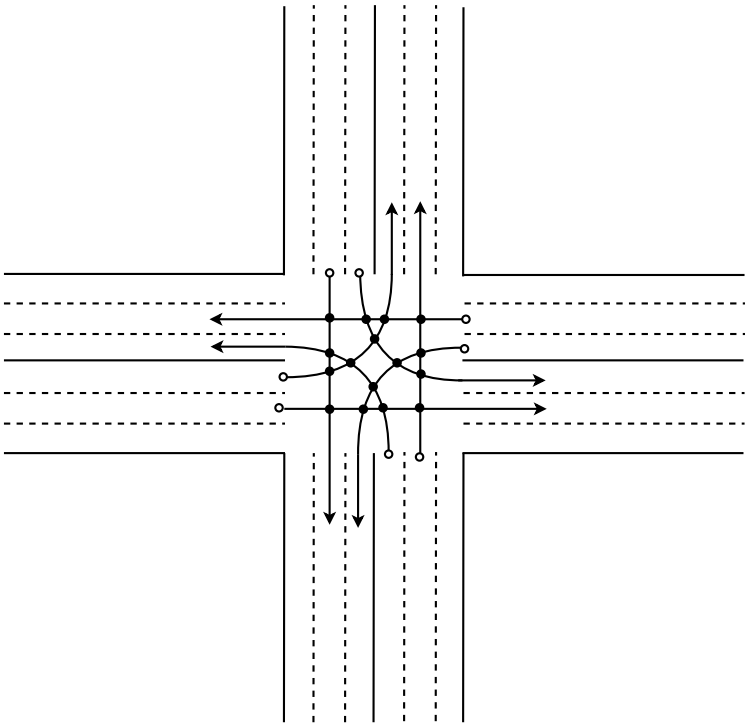


Figure 5.3: A four-way intersection

5.6 The HAIM Algorithm

The HAIM algorithm is a combination of four levels of heuristics each resolving some of the conflicts sequentially and ultimately having conflict-free scheduling of vehicles. The first heuristic in HAIM resolves conflicts of vehicles in the approach region and prevents rear-end collisions between two successive vehicles in the same lane. We call this heuristic as Lane Conflict Prevention Heuristic (LCPH). The LCPH will return a velocity that will satisfy these requirements; we call this velocity as the V_{lane}. The next three heuristics resolve intersection conflicts in a successive manner. The names of these heuristics in order of their application are i) First Enter First Serve Heuristic (FEFSH), ii) Window Heuristic (WH), and iii) Reservation Heuristic (RH). Each of these heuristic layers will return one velocity each named V_{fefs}, V_{window}, and $V_{reservation}$ respectively. These velocities may or may not have the same value. One of these three velocities will be set as the target velocity V_{target}. V_{target} will be evaluated using the selection logic defined later in this chapter.

Before we go further on to the HAIM algorithm, we should first discuss the data structure used by the IM to store reservation data corresponding to conflict points and depart points.

The data structure used for conflict point reservations data is called *Record* and for saving the single timing values such as *arriveTime*, we use the data structure named *Register*. The data structure *Register* resembles a simple register that stores one value of the double datatype. On the other hand, *Record* data structure resembles the Queue data structure with First In First Out property. Figure 5.4 shows a record of reservations for a conflict point. Each element of the record will have two fields corresponding to the *in_time* and *out_time* of the vehicle at that conflict point. Here the *in_time* and *out_time* are the times at which the vehicle will enter that conflict point and exit that conflict point respectively and they are calculated based on the V_{target} of the vehicle. The length of the record is decided based on the estimate of the number of vehicles in the scenario as the vehicle under consideration can only have a conflict with vehicles already in the scenario and not the ones that have already left or the ones still to enter. We take its size to be 20 i.e. at every conflict point, a record of the previous 20 reservations will be kept. The elements of the record are always kept in decreasing order of *in_time* i.e. the top of the record (having index 0) will have the largest absolute *in_time*. With every new reservation, one element will be added to the record. We can add an element at any position in the record. We will see later in the respective sections that FEFSH and RH only make reservations at the top of the record. Whereas, the WH can insert an element at any position in the record. Always an insertion in the record is followed by shifting of all the following elements by one position and the last element gets deleted. We will next describe each of these layers of heuristics in detail.

5.6.1 Lane Conflict Prevention Heuristic

The process of resolving conflicts starts with guaranteeing a safe traversal of the vehicle in its lane. Inside the lane, a vehicle can have collision only with either the vehicle ahead or the vehicle behind in the same lane. Collision free lane traversal can be realized if, for every vehicle, we resolve conflict only with the vehicle ahead in the same lane. The next incoming vehicle will also run this routine, as a result, we will have a conflict-free lane traversal throughout the scenario. Based on this logic, we design our approach towards resolving head-on collisions in the lane.

The strategy applied to prevent a head-on collision with the vehicle ahead in the lane is to make the departure time of the vehicle greater than the departure time of the previous vehicle

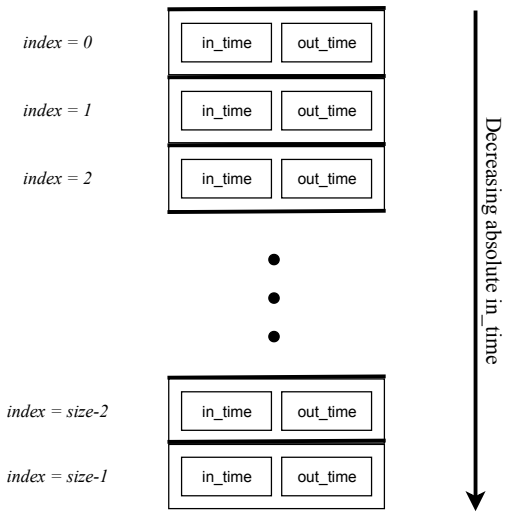


Figure 5.4: The Record data structure

in the lane plus a safety margin. We will next elaborate on this.

The departure time that we are talking about is the virtual departure time. The virtual departure time is the time at which the vehicle would reach the Virtual Departure Line if the vehicle continues to travel with V_{target} velocity after passing the IEL. However, the vehicle is actually required to accelerate to the *maxVel* after passing the IEL. As a result, the vehicle would never reach the VDL at the virtual departure time unless V_{target} is the same as *maxVel*. The concept of virtual depart time and virtual depart line is introduced to create enough separation between successive vehicles in the lane around the IEL so that there is no head-on collision in the depart region. The length of RDR is calculated using Newton's laws of motion and represents the maximum length required for the greatest velocity transition using the considered vehicle parameters.

We will call the virtual depart time as depart time only and VDL as the departure point or departure line. The IM has a depart time register for every lane and updates that register every time a vehicle on that lane is assigned its V_{target}. This means that the depart time register will contain the virtual depart time of the last vehicle that entered that lane.

Let us define a function called *findVelocity(vId, targetTime, targetDis, v_max)*. This function is present at the IM, and the IM makes use of this function whenever it needs to find a velocity that will enable a vehicle with vehicle identification number given by *vId*, to reach a point which is at a distance *targetDis* from the ASL at the time *targetTime* or later, starting

with the arrival velocity (*arriveVel*). Please note that the goal here is to reach the *targetDis* not before the *targetTime*. Also note that the IM has the arrival information of every vehicle in the scenario and can access them using the vehicle identification (*vId*). The algorithm for *findVelocity()* function is given in Algorithm 7.

Algorithm 7: $findVelocity()$ function

Input: *vId, targetTime, targetDis, v_max*
Output: vel

$t\_e = arriveTime[vId]$;
$t\_t = targetTime$;
$v\_e = arriveVel[vId]$;
$T = t\_t - t\_e$;
$D = targetDis$;
if $(t\_t < t\_e \,||\, (acc * T * (acc * T + 2 * v\_e) - 2 * acc * D) < 0)$ **then**
    vel = v_max ;
else if $(v\_e * T < D)$ **then**
    vel = $(v\_e - acc * T) - \sqrt{acc * T * (acc * T + 2 * v\_e) - 2 * acc * D}$;
else if $(v\_e * T > D)$ **then**
    vel = $(v\_e - acc * T) + \sqrt{acc * T * (acc * T + 2 * v\_e) - 2 * acc * D}$;
else
    vel = v_e;
end
vel = min(vel, v_max);
return *vel*

In Algorithm 7, the condition inside *if* can be read as "if either the target time is smaller than the enter time of vehicle or it is not physically possible for the vehicle to reach the target distance at the target time with the given acceleration". The second condition (after $||$) inside *if* can be rearranged as shown below.

$$D > v\_e * T + \frac{acc*T*T}{2}$$

Which encodes the same explanation given above. If it is physically possible for the vehicle to reach the target distance at the target time, the velocity will be given by the following expressions given inside the two *else if's*. These expressions are derived from Newton's laws of motion.

To obtain the V_{lane} for a new incoming vehicle with *vId* as its vehicle identification, the *findVelocity()* function will be called with the following arguments.

vId = vId

targetTime = lastVehicleDepartTime[laneNum]

targetDis = approachLen + intersectionPathLen + departLen

v_max = Maximum velocity allowed in scenario (taken as $17m/s$*)*

Where, *laneNum* is the identification of the lane on which the vehicle entered, *approachLen* is the approach length, *intersectionPathLen* is the length of the trajectory of the vehicle inside the intersection, and *departLen* is the depart length of the scenario.
V_{lane} obtained above is saved as a vehicle information with the variable name *laneVel* and can be accessed later using the vehicle id.

We will be using the *findVelocity()* function further in this chapter inside the FEFS heuristic and the Reservation heuristics as well. As for the Buffer region, as previously mentioned, the velocity of the vehicle is limited by the velocity of the vehicle ahead in the same lane inside the buffer, thus, head-on collision inside the buffer is prevented given that the vehicle made a stable entry into the buffer i.e. it was not about to collide at the time of entering. The velocity returned by the *findVelocity()* function will at this stage will be called as the V_{lane}. V_{lane} is the maximum velocity with which the vehicle can travel and not have any collision in the lane. This means, the velocities returned by the three heuristics for resolving intersection conflicts are always less than or equal to the V_{lane} or in other words, V_{lane} is the limiting velocity for the three intersection conflict resolving heuristics.
For right turning vehicles, the V_{lane} will be their final velocity as they do not have any intersection conflicts. The maximum velocity for right turning vehicles can have a lower value than other vehicles to have a safe turning at the intersection.

5.6.2 FEFS Heuristic

This is the first heuristic to resolve the intersection conflicts. The name FEFS stands for First Enter First Serve and this heuristic applies the same principle in its working. The basic scheme in the FEFS Heuristic is to reserve conflict points for vehicles in order of their arrival into the scenario. We have seen earlier that there are 4 conflict points in each vehicle's trip through the intersection and the reservation data for these conflict points are stored in the form of records at the intersection manager. To resolve conflict, FEFS heuristic finds a velocity that will result in crossing times at the first two conflict points in the vehicle's path such that they are greater than the previous reservations at those conflicts points. In other

words, the FEFS scheme makes a reservation only at the top of the reservation record i.e. after all the existing reservations.

To calculate V_{fefs}, we use the same procedure that we used to obtain V_{lane} except that here we apply it twice for each of the first two conflict points and the target time for each of these procedures will be the depart time at the top of the record of the respective conflict point plus the safety gap. The maximum value of V_{fefs} for any vehicle is limited by its V_{lane}. V_{fefs} will be the minimum of the velocities corresponding to the first two conflict points. The reason only the first two CPs are considered in the FEFS scheme is that only a small fraction of vehicles have a conflict on the later two conflict points. Let us see why is it so. Figure 5.5 shows all the conflict points present inside the intersection region. We can see in this figure that every conflict point is an intersection of two lane-to-lane connections. The indexing of these conflict points shown in this figure is done using the position of that conflict point in both the lane to lane connections. For instance, if the index of any conflict point is *(a, b)*, it means that it is a^{th} conflict point in one of the lane connective and b^{th} conflict point in the other lane connective with the smaller number coming ahead.

On observing, we can see that all these indices are one of the following possibilities.

(1, 3), (1, 4), (2, 3), and (2, 4)

This means that the conflict point under consideration is always either the first or the second conflict point in one of the lane to lane connective and either the third or the fourth conflict point in the other lane to lane connective.

Now according to the FEFS heuristic, where only the first two conflict points in a lane to lane connections are checked for conflicts we can call the lane to lane connection that has the lower index as *considerate* and the lane to lane connection that has the higher index as *non-considerate*. With the possible indexing, we can say that at every conflict point, always, one of the lane-to-lane connections is *considerate* and the other is *non-considerate*. Put in other words, a lane-to-lane connection will be *considerate* at the first two conflict points and *non-considerate* at the later two conflict points. As per this strategy, at every conflict point, one of the two lane-to-lane connections will always be considerate. This is the reason that most of the conflicts are resolved by the FEFS Heuristic. However, still, some conflicts remain unresolved. The reason behind these remaining conflicts is that in the FEFS

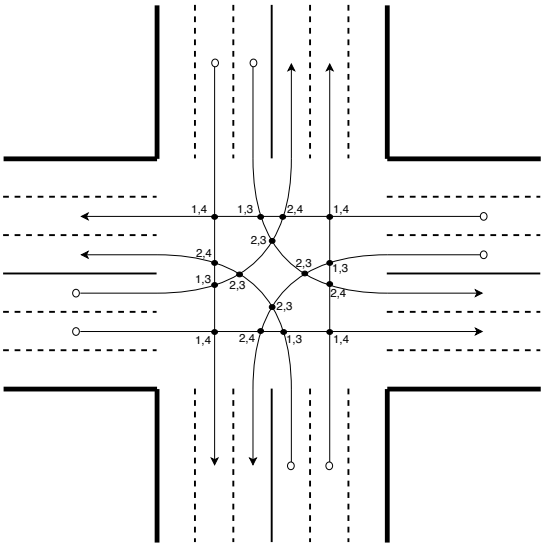


Figure 5.5: Conflict point indexing

heuristic, pre-existing reservations at the third or fourth conflict points are not checked and this can result in a collision if a slow-moving vehicle that entered earlier has a reservation on either the third or the fourth conflict point at the same time. In other words, any vehicle that shares either the third or the fourth conflict with the vehicle under consideration entered before and already has a reservation at the shared conflict point at an overlapping time slot then there will be a collision.

The procedure to obtain the V_{fefs} involves calling the *findVelocity()* twice. Once for the first conflict point and then for the second conflict point. The arguments passed to the *findVelocity()* are as shown below.

vId = vId; vehicle id of the vehicle under consideration
targetDis = distance of the concerned conflict point from the ASL
maxVel = laneVel[vId] i.e. the laneVel calculated for that vehicle

targetTime = cp_record[0]

The pseudo-code for finding the V_{fefs} is shown below.

```
v_cp1 = findVelocity(v_id, target_time_cp1, target_distance_cp1, laneVel[vId])
v_cp2 = findVelocity(v_id, target_time_cp2, target_distance_cp2, laneVel[vId])

fefsVel = min(v_cp1, v_cp2, laneVel[vId])
```

The minimum of the velocities obtained for the two conflict points and the V_{lane} for that vehicle is set as the V_{fefs}. The reason behind taking the minimum of the two velocities corresponding to the two conflict points is that the requirement in the *findVelocity()* function is that the returned velocity should result in a passing time not less than the target time. So, taking the minimum of the two velocities will ensure that this condition is respected at both the conflict points. And if both the velocities are greater than the V_{lane}, then V_{lane} is set as the V_{fefs}. After calculating V_{fefs}, it is saved as *fefsVel* at the IM.

5.6.3 Window Heuristic

So far we have resolved the conflicts in lane preventing head-on collisions in vehicles in the same lane using the lane collision prevention heuristic and also all the conflicts occurring at the first two conflict points in the path of any vehicle. Though most of the conflicts are resolved by the FEFS heuristic, there remains a possibility of conflict and collision at the last two conflict points as they were not checked in the FEFS heuristic. The strategy while finding both the V_{lane} as well as V_{fefs} has been to make the vehicle pass after the passing time of the previous vehicle at that point. Both these strategies makes use of the *findVelocity()* function explained earlier. Also, both these strategies made use of only one register that saves the passing time of the previous vehicle. In lane velocity calculation, this register was the previous vehicle departure time register for the corresponding lane and in the V_{fefs} calculation, this register was the top element of the reservation record of the corresponding conflict point.

The strategy employed to calculate the V_{fefs} suffers from another drawback except for some unresolved conflicts. This drawback is the phenomenon of *piling of delays*. The piling of delays happens when the delay in the trip of one vehicle gets added to the trip of another vehicle and this happens successively to generate large delays. This happens at large traffic volumes when the vehicles are scheduled only using the FEFS heuristic (by neglecting the colliding vehicles for a while). At lower traffic, when there is sufficient duration between the arrival of vehicles, and only a few vehicles are required to reduce their speed to meet the passing condition at the first two conflict points. In such conditions, trips of vehicles are not much dependent on other vehicle's trip. In such low traffic conditions, delays in vehicle trips do not pile up. However, at higher traffic when the duration between the arrival of vehicles

is very small, the trips of vehicles are more dependent on vehicles that entered earlier. Thus the delay in the earlier vehicle's trip will get added to their trip as well.

To prevent this, we need to have a mechanism that can break this delay pile and not allow them to accumulate over time. This can be done if vehicles are allowed to pass through the intersection before the previously entered vehicle. This will prevent unnecessary slowing down of vehicles that can pass the intersection with a faster velocity. Also, the dependency of vehicles on previously entered vehicles will decrease resulting in better efficiency of the intersection control.

This is where the Window heuristic comes into the picture. In the window scheme, rather than adding the reservation only at the top of the reservation record, the entire record is scanned for a window in between reservations at all four conflict points. The greatest velocity for which such a window exists will be selected as the V_{window}. To find the V_{window}, we use an iterative strategy. The iterations start with V_{lane} for that vehicle and go in the decrement mode. At each iteration, a window is checked in the complete reservation record at all four conflict points. If a window is found, iterations stop otherwise it will continue with another decrement. Iterations will stop if the iterative velocity reaches zero. That would mean that no window is present. In that case, the V_{window} will be set as zero. The algorithm to calculate V_{window} is shown in Algorithm 8.

The function $isWindowAvailable(cp_i, vel, d\_cp_i)$ searches entire record of the given conflict point (cp_i) which is at a distance d_cp_i from the ASL to search whether the travel time corresponding to the given velocity (*vel*) falls inside a window or not. It returns true when a window is present for the given velocity in the reservation record of all four conflict points; otherwise, it returns false. If the velocity returned by the window scheme is non zero, it is always conflict-free as it is derived by considering previous reservations at all four conflict points.

5.6.4 Reservation Heuristic

Window heuristic resolves some of the conflicts that are left unresolved in the FEFS heuristic but it is not guaranteed that it will resolve all those unresolved conflicts. This is because the Window heuristic may not get a window in between previous reservations and can return a zero velocity. This means the possibility of conflicts has still not vanished. To

Algorithm 8: Function to find V_{window}

Input: *vId, cp1, cp2, cp3, cp4, d_cp1, d_cp2, d_cp3, d_cp4*
Output: V_{window}

$vel = laneVel[vId]$;
$cp1\_clear = False$;
$cp2\_clear = False$;
$cp3\_clear = False$;
$cp4\_clear = False$;
while $(vel > 0$ ***and not***$(cp1\_clear$ ***and*** $cp2\_clear$ ***and*** $cp3\_clear$ ***and*** $cp4\_clear))$ **do**
    $cp1\_clear = isWindowAvailable(cp1, vel, d\_cp1)$;
    $cp2\_clear = isWindowAvailable(cp2, vel, d\_cp2)$;
    $cp3\_clear = isWindowAvailable(cp3, vel, d\_cp3)$;
    $cp4\_clear = isWindowAvailable(cp4, vel, d\_cp4)$;
    vel = vel - del_v;
end
if *vel < 0* **then**
    vel = 0;
    else
        vel = vel + del_v;
    end
end
$V_{window} = vel$;
return V_{window}

resolve the remaining conflict cases and making the overall algorithm free from collisions, the third heuristic is needed. We call this as the Reservation heuristic.

In Reservation Heuristic, the last two conflict points are resolved for conflicts. A procedure same as that of the FEFS heuristic is employed but for the later two conflict points instead of the first two. We again call the *findVelocity()* function twice for the last two conflict points and assign the minimum velocity out of two and that vehicle's V_{fefs} as the $V_{reservation}$. The following pseudo-code depicts the logic of the Reservation heuristic.

```
v_cp3 = findVelocity(vId, targetTime_cp3, targetDis_cp3, laneVel[vId])
v_cp4 = findVelocity(vId, targetTime_cp4, targetDis_cp4, laneVel[vId])

reservationVel = min(v_cp3, v_cp4, fefsVel[v_id])
```

5.6.5 Decision Logic

Figure 5.6 shows the decision flow of the HAIM algorithm. First three heuristics are used for all vehicles i.e. the calculation of *laneVel, fefsVel* and *widnowVel* is common for all, whereas the $V_{reservation}$ is calculated only for those cases whereV_{fefs} and V_{window} could not resolve conflicts on the last two conflict points. Whether conflicts are resolved by V_{fefs} or not is determined using internal simulation. In the internal simulation, the arriving time of the vehicle is calculated using V_{fefs} on the third and the fourth conflict points. Introduction to the simulator used for this purpose is given in the Section 5.7. If there is an overlap with earlier reservations, then it would mean that there is an unresolved conflict.
After V_{fefs}, the V_{window} is calculated for all vehicles. If V_{window} is nonzero, then the algorithm will be terminated and V_{window} will be set as V_{target}. This is because V_{window} gives a reservation that is earlier to some of the previously entered vehicle hence the obtained velocity is greater than what we would have obtained using the FEFS or the Reservation heuristic. Also, if V_{window} is nonzero, it will never result in a conflict. So, in such a case, we need not calculate $V_{reservation}$. In case V_{fefs} results in a conflict and V_{window} is not nonzero then, $V_{reservation}$ is calculated and set as V_{target}.

5.7 Simulation

The presented algorithm (and scenarios) is realized and simulated in the SUMO traffic simulator. SUMO (Simulation of Urban MObility) is an open-source, highly portable, microscopic road traffic simulation tool. In SUMO, a scenario is built by combining various input files containing specifications of nodes, edges, connections, routes, and network [120]. All these files are combined in a configuration file to start the simulation. TraCI (Traffic Control Interface) library is used to control vehicles in the simulation. A screenshot of the simulation is given in Figure 5.7

Along with the proposed heuristic, we have also implemented, on the same platform, two more autonomous intersection control algorithms along with the conventional traffic light control. The first one is the FCFS reservation policy presented in [62] with intersection modeled as a group of square blocks and these blocks are reserved for incoming vehicles

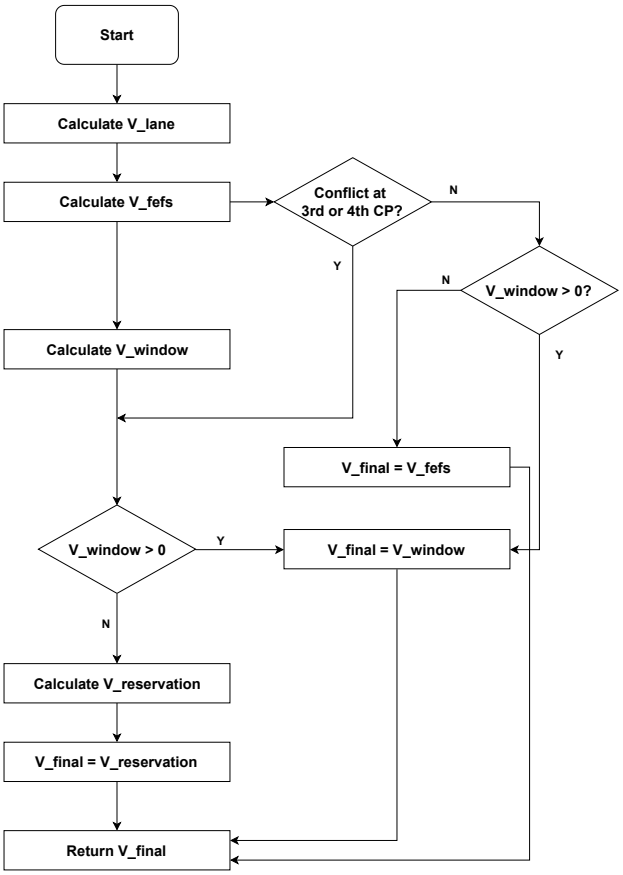


Figure 5.6: Decision flow of the HAIM algorithm

Table 5.2: Parameter Values

Parameter	**Value**
Maximum Velocity	$17m/s$
Acceleration	$3m/s^2$
Safety gap	$500ms$
Approach length	$150m$
Buffer length	$50m$
Simulation time	$3600s$

depending on their time-line. A granularity of 48 is taken, which means, the intersection is modeled as a group of 48x48 square blocks.

The second scheme implemented is the CIVIC algorithm [84]. The CIVIC algorithm solves the intersection management problem using nonlinear constrained optimization by minimizing the length of overlap of trajectories of vehicles inside the intersection area. The length of the overlapping trajectories is calculated by using Newton's equations of motion and the ob-

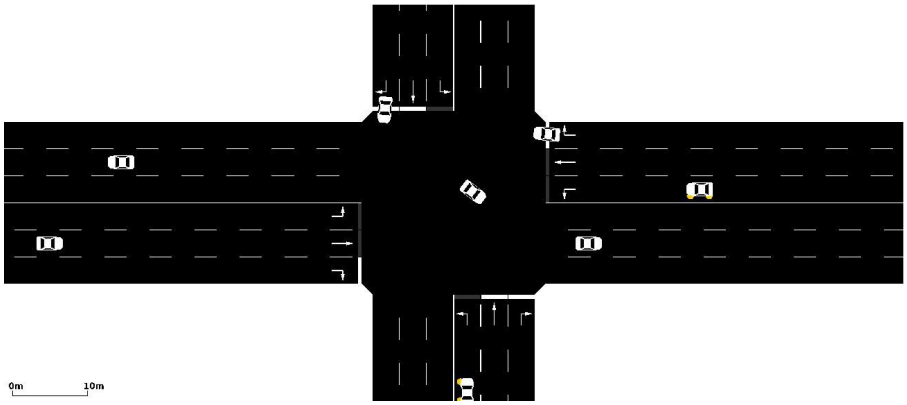


Figure 5.7: A simulation screenshot

jective function is calculated by adding these overlaps for each pair of vehicles on conflicting lanes. Also, the phase conflict map in [84] is modified according to the phase numbers used in the intersection model.

These two algorithms are very popular in the autonomous intersection management domain and can be considered as benchmark algorithm also, they belong to two different approaches towards autonomous intersection management namely multi-agent approach and nonlinear constrained optimization approach respectively. For these reasons, we choose them for making comparative analysis with the proposed heuristic approach. In all the implementations, we have used the same intersection, traffic, and safety gaps. In the proposed scheme, we adapt the vehicle length in simulation by adding a delay equal to the time that vehicle would take to pass the conflict point and depart point. Since vehicles travel only on the center of their respective lane, a lateral safety gap is not required. Results are obtained for the delay in vehicle trip time. Delay time is chosen as the parameter for performance evaluation because it is an independent parameter and other performance matrices such as the mean velocity of vehicles etc. will have non-orthogonal dependence on it. The values of parameters used in the simulation are given in Table 5.2.

Simulation is performed for different traffic densities for a simulation time of 3600 seconds. With similar environmental and traffic conditions, delay times are recorded for different schemes of intersection management. These are shown in Table 5.3 and visualized in Figure 5.8.

As we can see, the presented algorithm outperforms the other three intersection management algorithms. Traffic lights have the largest delay owing to the fixed halt of

vehicles during red light making vehicles decelerate and again accelerate on green light causing a delay in these transitions of velocity plus the wait time during the red phase. Our scheme outperforms the FCFS algorithm because, in the FCFS algorithm, an availability check is performed at all the granules (blocks) which are lying in the trajectory of the vehicles. On the other hand, in our scheme, we only need to make 4 availability checks corresponding to the four conflict points. And the reason why the proposed algorithm results in less delay than the CIVIC algorithm is that in the CIVIC algorithm, optimization is performed only to minimize the overlap of trajectories of two vehicles inside the intersection area, and no factor is added to this optimization that targets to reduce the delay in scheduling.

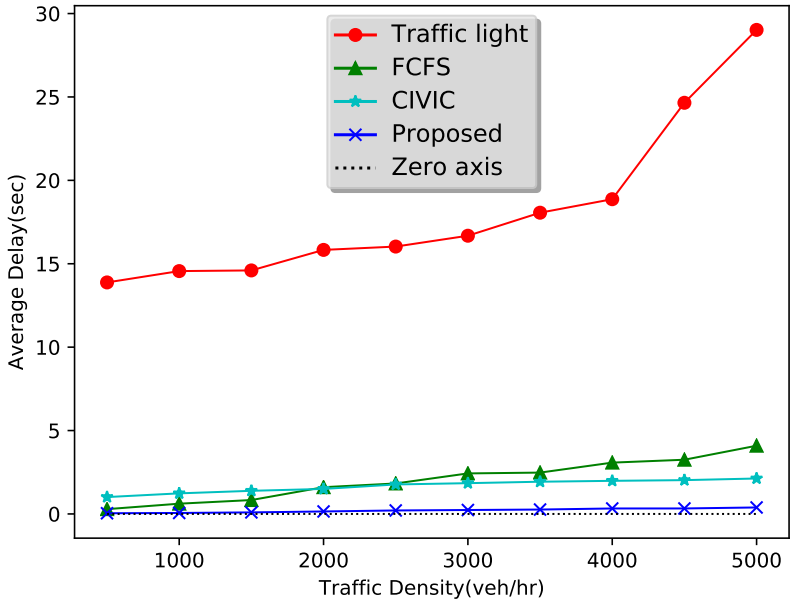


Figure 5.8: Simulation results

5.8 Discussion

The HAIM algorithm presented in this chapter had approached the autonomous intersection management task with a heuristic algorithm. The reason for choosing a heuristic approach is to keep the computational requirements to a minimum. This will facilitate the algorithm in becoming a suitable method for the autonomous intersection management task owing to the

Table 5.3: Simulation results

	Delay (in Sec.)			
Density(v/h)	**TL**	**FCFS**	**CIVIC**	**Proposed**
500	13.88	0.288	1.012	0.0416
1000	14.56	0.615	1.234	0.059
1500	14.60	0.833	1.387	0.093
2000	15.83	1.604	1.500	0.149
2500	16.03	1.827	1.768	0.209
3000	16.68	2.430	1.845	0.236
3500	18.06	2.479	1.932	0.264
4000	18.87	3.077	1.985	0.328
4500	24.65	3.250	2.027	0.329
5000	29.02	4.090	2.125	0.389

real-time behavior of the problem. All the four heuristics have a complexity of the order of *O(k)*, where *k* is a constant, i.e. each heuristic will take a constant amount of time making the overall complexity of the intersection management of *n* vehicles to be of the order *O(n)* as each vehicle will be considered once. This is a very significant advantage as the real-time restrictions are more sensitive for the intersection management problem compared to the requirement of minimizing delay in trip time and maximizing the efficiency of the intersection management.

Three heuristics out of four make use of an analytical formula for the computation of their velocity. Whereas in Window heuristic, iterations are performed for searching windows at all four conflict points in between successive reservations. As there are four conflict points, ten reservation fields in each reservation record, and the iterative velocity decrements by 0.1 m/s for a velocity span of 17m/s, for calculating V_{window} for one vehicle, the maximum computational time will be proportional to (4*20*17/0.1) which will still result in a complexity of the order of *O(k)*.

In our approach to find the maximum safe velocity for passing the intersection, we first try to find the maximum velocity that will not result in a lane conflict. This velocity will be called the V_{lane} for that vehicle. Since any velocity greater than V_{lane} can result in a collision, the V_{lane} sets a limit for the next three heuristics.

After resolving lane conflicts, we shift our focus to resolving intersection conflicts. Using the restrictions on the lane behavior of vehicles and trajectory inside intersections, we reduce

the collision space to some regions of the intersection region called Conflict Points. There are a total of 16 conflict points with 4 conflict points in each vehicle's trip. We can also say that the intersection management problem of the 4-way intersection has been reduced to that of 16 single lane intersections. In FEFS heuristic, we resolve conflicts only for the first two conflict points. The FEFS heuristic resolves the majority of conflicts at lower traffic density. At lower traffic density the duration between the arrival of successive vehicles is large, thus only a few vehicles have to insert a delay in their trip to respect previous reservations. However, at higher traffic, the trip of a greater number of vehicles overlap with reservations made at conflict points. These vehicles will require to slow down thus introducing a delay in their trip. Since there are shared conflict points, delay in one vehicle trip can get added to other vehicle's trip. In FEFS heuristic, vehicles are granted passing only after the previous vehicle's schedule. Thus, delays can get piled up to result in large delays in scheduling. A mechanism is thus required to break this delay pile before it affects the performance of intersection management.

The next layer of heuristic serves this exact purpose. It allows vehicles to get reservations even before earlier entered vehicles. In this heuristic, a window is searched in between reservations at all the conflict points. For this reason, this is called Window heuristic. Though window heuristic breaks the delay pile and returns conflict-free velocity when it returns a velocity, it is not guaranteed to always return a velocity. There might be a case where the FEFS heuristic results in conflict at either of the last two conflict points and the Window heuristic could not find a window. In such a case, we will require the final heuristic layer called the Reservation heuristic. This heuristic will resolve conflict at third and fourth conflict points and we will ultimately have conflict-free intersection management.

As we see, each layer in HAIM serves a specific purpose. The FEFS heuristic finds velocity for most of the vehicles at low traffic. As traffic increases, the significance of the Window heuristic will increase. It will break delay piles and enable efficient control of the intersection and finally, the reservation scheme will guarantee no collision condition.

The simulation result obtained shows that along with satisfying safety requirements, the HAIM algorithm also performs better in terms of delay induced in vehicles trip. We performed a comparative study with two other approaches towards intersection management. One belongs to the multi-agent-based approach and the other belongs to the nonlinear

constrained optimization-based approach. The HAIM algorithm outperforms both these schemes and thus proves out to be more efficient.

As we mentioned earlier, through the buffer region, we have abstracted away from communication and computation related delays. We can adjust the length of the buffer region to account for the unreliability in communication and computation. Increasing buffer length will not create any issue until the approach length is kept sufficiently long to have velocity transitions. We had earlier discussed how a constant velocity based intersection management results in lower emissions, lower energy wastage, stop-less intersection passing, and more comfortable journey for passengers. These benefits along with the encouraging results in favor of the HAIM algorithm make the HAIM algorithm a very efficient and safe intersection management algorithm.

5.9 Summary

In this chapter, we presented the HAIM algorithm for the intersection management of autonomous vehicles. The complexity of the task is first reduced by defining the lane behavior and turning trajectory for vehicles to scheduling vehicles at four conflict points. To resolve all the conflicts, a four-leveled heuristic is employed in which the first heuristic resolved all conflicts that could result in rear-end collisions whereas, the next three heuristics resolved all the intersection conflicts successively. The HAIM algorithm, along with being free from computationally intensive procedure implements a velocity-based intersection management procedure in which vehicles are required to attain and then maintain the velocity assigned by the centralized intersection manager. The advantages of the fixed velocity-based approach are also discussed. A comparative analysis of the HAIM algorithm with three other intersection management schemes is performed. Results show that the HAIM algorithm outperforms all other intersection management schemes with respect to the delay caused in scheduling.

In the next chapter, we will present the literature pertaining to the formal verification, their application to autonomous systems such as autonomous robots and autonomous vehicles and finally we will see the literature that targets the verification using statistical model checking technique.

Chapter 6

Formal Verification Literature Survey

6.1 Formal Verification

Formal verification has now become a well known technique in the industrial domain. It's popularity has grown to a level that in various industrial applications, verification is considered as a very crucial part of the product development life cycle. Typically, the verification and validation phase takes most of the development time [121]. The importance of formal verification comes from the fact that it is based on solid mathematical foundation and gives the maximum coverage of state space with respect to other validation techniques. With the rise in the use cases, verification techniques have also evolved to expand the classes of systems they can be applied on. The field of formal verification has not only grown in length but also in width. By length we mean the classes of systems on which these techniques can be applied and by width we mean the options or different underlying formalisms or languages that can be used for this purpose. There are several available formalisms to: (a) model the system and (b) specifying the properties. However, the choice of these two formalisms are sometimes interdependent. A very detailed and precise introduction to the formal verification is given in [23]. The author in this work give introduction to the underlying concepts, techniques, tools and formalisms. Though this article do not cover all the tools and formalisms available today as it dates back to the time when formal verification was not in as advanced stage as it is today but still it is a very good read for interested readers.

A much recent article that give very good insight about the verification methods that are in usage today is given in [122]. In this work, authors discuss the relationship between verifica-

tion and logical reasoning and also provide an overview of all the current popular tools and methods that are there at the current time.

The benefits of using formal verification has attracted its use in a spectrum of applications. In various applications, the use of formal verification has now become indispensable such as in designing of digital systems, communication protocols, operating systems, etc. Whereas in various other applications such as cryptographic encoding, block chain smart contracts, etc. its usefulness has been realized and its use is becoming more important than ever.

Since in this thesis we are dealing with the formal verification of the HAIM algorithm, which is an intersection management algorithm for autonomous vehicles, we will restrict our discussion to the literature associated with formal verification of autonomous systems such as autonomous robotic systems and/or autonomous vehicles only.

6.1.1 Formal Verification of Autonomous Systems

Autonomous systems is a class of systems that covers a wide range of application scenarios. These systems often work in complex environments and have minimum input from humans. They are mainly characterized by their decision making ability using the gathered knowledge of the environment and objects inside it. Some examples of application scenarios of autonomous systems are industries (eg. autonomous warehouses), transportation (eg. autonomous vehicles), inside house (eg. robotic assistants) and even in space (eg. mars rover). Increasing popularity of autonomous systems and their application in more and more human interactive scenarios have made the formal verification of such systems a critically important procedure as it may help in giving safety related assurance along with those related to performance and efficiency.

A comprehensive study on current state-of-the-art for formal modeling, specification and verification of autonomous systems is discussed in [123]. In [124], and [115] authors emphasize on the fact that the system development life cycle should be considerate of detecting the faults in the design as early as possible. They propose a workflow that incorporates a incremental design process by using a component based design language and verification framework that allows building up the reasoning on the behavior of the overall system starting from the component level.

The motion planning of a group of autonomous robots in a given environment is a well

known research problem. This can include simple tasks such as "Go from point A to B" but it can well be quite complex also such as "From the second street, navigate through the roundabout and join the main street and then park at the central parking". Though the temporal logic such as CTL and LTL are capable of expressing such rich requirements, a challenge that remains is to construct a finite model that can faithfully capture the robot motion and control capabilities. However, the added challenge in formally verifying the functioning of such systems is to prevent state-space explosion that may arise even in small systems. Authors in [75] use a distributed formal synthesis approach to deal with this problem. They proposed a framework that allows for using the given requirements in Regular Expressions to automatically synthesize the control and communication strategies for robots.

In [67], authors demonstrate modeling and formal verification of heterogeneous multi-agent system using three case studies. In [76] and [79], authors approach the verification of the robotic swarm algorithms using model checking. The robotic swarm is a group of individual robots which works in unison to fulfill some task. An example of requirement from such systems is that is should be fail-safe i.e. failure of any robots in this swarm should not result in the failure of the complete swarm. To check the swarm algorithms for such requirements, either the swarm would have to be realized in actual or in simulation. Both however, will keep the analysis relevant only to the particular scenario. Keeping this in mind, authors chose to perform formal temporal verification of the swarming algorithm using model checking.

A popular representation of autonomous systems is through *agent-based* notation. In agent-based representation of autonomous systems, the autonomous entity is called as an agent and an agent is characterized by the ability to make the decision of when to act and how to act in response to the changes in its environment. This concept of agent came in late 1980's and has been used vastly in both industry and academia. However, with the further development of this line of research, it was realized that these agents are required to have explanation of *why* it takes a certain course of action along with *when* and *how*. This extension of agents use the architecture given by the *Belief, Desire,* and *Intent* (BDI) approach and are known as *Rational Agents*. Article [125] gives a very good introduction to rational agents and also discusses how we can approach formal verification of autonomous agents represented as rational agents. In [126], authors perform formal verification of the high level decision making software components of an autonomous vehicle using rational agents which are presented

as being intelligent instead of reactive, and has an ability to assess the situation to make the best decision. For instance, which obstacle to collide with to keep the damages to a minimum in case a collision is unavoidable. Rational agents are modeled using Gwendolen agent modeling language, and AJPF (Agent Java Path Finder) is used for formal verification of specifications written in Linear Temporal Logic.
In [73], authors tackle the problem of formally verifying a decentralized self-adapting system using model checking. They first perform the behavior analysis of the design of the system under consideration which is a decentralized traffic monitoring system, then perform model-based testing of the implementation of the system and finally run-time diagnosis after system deployment. In [127], authors undertake the task of designing a high level planner/scheduler for Care-O-bot, a robotic assistant and then formally verifying it using model checking.
The article [128] is an attempt in the direction of analysing the available options with respect to the choice of formalism and level of formality for the formal analysis of autonomous vehicle systems. They perform a case study over the formal verification of the Lateral State Manager module of an autonomous vehicle using three different verification approaches namely Supervisory Control Theory, Model Checking and Deductive Verification. The goal is not to compare rather differentiating based on the objective of technique and studying how multiple formalisms can help to deal with challenges in developing the autonomous vehicle technology.

In literature a work that talks about the formal verification of an autonomous intersection management algorithm which is indeed the work closest to our work as presented in the next chapter is given in [129], where the authors approach the safety verification of an intersection using the KeYmaera theorem prover. They verify the safety property of the two most basic building blocks of any intersection scenario namely T-intersection/merging and two-lane intersection. The properties verified corresponding to these two cases in this work translate to: i) If the vehicle and stoplight start in a controllable state, then the vehicle will never enter the intersection while the light is red and ii) If the stoplight and the two cars start in a controllable state, no car will enter the intersection while its light is red, respectively. Our work differs from this work in two respects, which are: (i) We present verification of the autonomous intersection management algorithm as applied on a four-way intersection with 3 lanes in either direction and (ii) We use model-theoretic verification because the model of

the system involves much realistic situations such as dynamic and non-deterministic instantiation of vehicles, custom data structure and layered nature of the heuristic which makes model checking the preferred choice because of its ability to perform precise modeling of such system which would otherwise be complex and error-prone in a theorem prover such as KeYmaera.

In [130], authors introduce a spatial logic called Multi-Lane Spatial Logic (MLSL) and an abstract model of the multi-lane motorway based on the local view of the cars. Using the MLSL, properties needed for safety proof can be formulated and later used as guards and invariants in the design of abstract lane-change controllers. This work has been extended in [131, 132].Furthermore, [133] addresses how to model and verify traffic scenario such as intersection, turns, crossroads and T-intersections. This work is similar to the work presented with respect to the safety proof approach. The key idea behind the safety proof in both is to prove that the vehicle occupy and reserve disjoint spaces. However, the difference lies in the fact that in the mentioned work, the occupancy is determined using the view of each individual car whereas, a discrete grid model is used in the presented work and this grid is updated centrally by the Intersection Manager. The discrete position of a vehicle is determined in terms of cells of grid occupied by it at any moment. Also, since vehicles travel only in their respective lanes, this grid is one dimensional (array) and each route has a corresponding occupancy array in the lane and the intersection region.

6.1.2 Statistical Model Checking

Statistical Model Checking (SMC) is the extension of model checking for stochastic systems where the quantitative properties of the system are expressed in terms of measure of executions of the system satisfying certain temporal properties. The key idea behind SMC is to observe a fixed number of executions of the simulatable model of the system by certain monitoring procedure and deduce whether the system satisfies the desired temporal property or not. The results of SMC are generally associated with a bound of making the error and this is the trade off that has to be done to gain the advantage in terms of memory and time requirements [34]. SMC only requires the system to be simulatable [134] thus, increasing the class of systems that it can be applied on. We are going to discuss next some previous works that have used statistical model checking to verify quantitative measure of the

satisfaction of the required properties by any autonomous vehicle system. Reference [135] targets the problem of formal verification of autonomous systems with a case study on the traffic sign recognition in autonomous vehicles. They define the architecture of the system in a domain specific architectural language called EAST-ADL. This model is included with functional and non-functional properties such as time and energy constraints. To include the stochastic nature, the probabilistic extension of EAST-ADL constraints is defined and its semantics are translated to Uppaal-SMC for formal verification. In Reference [136], authors propose a verification architecture for automated cyber-physical systems. They perform two case studies corresponding to perception system and decision making system. Their main idea is to formulate some Key Performing Index (KPI) in a temporal language and use them to guide the verification using Statistical model checking. In another work [137], the authors present verification of the functioning of controllers present in an autonomous vehicle to prevent collision in a traffic jam situation. There are two types of controllers considered: first one is only responsible for following the front vehicle without collision and the other controller has responsibility of safe changing of lane. The controllers are modeled using C++ codes and the driveway is modeled as stochastic high-level Petrinets.

6.2 Summary

In this chapter, we presented the existing works targeting the formal verification of autonomous systems and associated algorithms. We observed that in the literature survey presented above, the focus is kept primarily on one or more sub-modules of autonomous vehicles that are responsible for any dynamic driving task of the vehicle such as lane following, lane change, and so forth. This means that the current literature on the application of statistical model checking in Intelligent Transport Systems (ITS) domain has primarily been vehicle-centric. To the best of our knowledge there has not been any work in the literature that deals with the statistical verification of any traffic management algorithm such as an intersection management while considering the dynamic nature of the problem and at the same time assuring the correctness of the modeling performed.

Inspiring from this, we introduce the application of SMC to ITS algorithms such as inter-

section management. In the next chapter, verification of the HAIM algorithm is performed at different stages of its implementation. Internal verification and artificial error injection testing is done to verify correct modeling of the system.

Chapter 7

Formal Verification of HAIM

7.1 Introduction

In the last two chapters, we have learned about the underlying techniques of formal verification and existing literature on the formal verification of autonomous systems such as autonomous robots, autonomous vehicles, domestic assistance robots, etc. using non-stochastic and stochastic techniques. We also mentioned that the statistical analysis of a safety property of any intelligent transportation system has not been carried out in the literature so far to the best of our knowledge. We believe that there is a genuine application of statistical techniques in this domain. Vehicular traffic in an ITS domain generally have a non-deterministic distribution thus, taking this into consideration while analyzing the performance of any ITS related controller should result in more faithful results. This motivates us to propose a statistical model checking of the HAIM algorithm. Along with providing the correctness proof of the HAIM algorithm with respect to the no collision property, we also give evidence of the faithful modeling of the overall system using two complementary procedures which are implementation verification and error-injection testing. In this chapter we will give every detail of the procedure of generating the correctness proof of the HAIM algorithm against no collision property using statistical model checking. The tool we have used for this purpose is the Uppaal model checker, particularly its SMC variant known as Uppaal-SMC.

In the following section, we give an overview on: challenges in formal verification of autonomous systems, choice of verification paradigm used and the choice of tools used in our work.

7.1.1 Associated Challenges

For every technology, no matter how efficient or time saving it is, it has to satisfy the most important property and that property is safety. Safety is of utmost importance in the technologies that interface with humans. That is the reason before deploying any technology it becomes mandatory to prove its safety. In fact, the U.S. Department of Transportation (DoT), in its latest guideline issue [14] for the autonomous vehicle development, has put great emphasis on verification, validation, and compliance of safety standards such as ISO 26262, IEC 61508, and so forth.

As a method to bring confidence in the safety of their technology, various autonomous vehicle original equipment manufacturers (OEMs) use their technology to drive for long distances in test drives on public roads and later use this driving distance as a measure of the confidence in the safety of their technology. This is known as public road testing. Although testing is an important step in any product development life cycle, it also has its limitations. In case of autonomous vehicle technology, the extent of testing that is required to guarantee with the given confidence, the safety of operation is impractical. The study presented in Reference [138] says that to prove with 95 percent confidence that self-driving fleet has a 20 percent lower fatality rate than that for human driven, it would require 100 vehicles to drive around 24/7 for around 225 years! This calls for a complementary technique in the development cycle that can generate correctness proofs for the corresponding subsystems in a reasonable amount of time. Formal verification is one such technique.

7.1.2 The Choice of Verification Technique and Formalism Used

Formal verification is a systematic approach that uses mathematical reasoning to verify that the specification (requirement) is preserved in the implementation (system model). As we discussed earlier, broadly there are two classes of formal verification methods which are: (i) Property-oriented verification and (ii) Model-oriented verification and in our case, model-oriented approach is more appropriate as we are not working with general properties, mathematical axioms, rather with we are completely aware of the system. In addition to that, the model-oriented tools present provide some features that can realize systems that are non-trivial otherwise. For instance, we have used dynamic instantiation of a vehicle template for

non-deterministic arrival of vehicles, just as real world traffic.

Verification is a crucial part of any hardware or software system development life cycle. It is aimed towards finding errors in the design as early in the development life cycle as possible. The process of formal verification starts with the representation of the system in a formal language that is suitable for the verification technique used. The choice of formalism to model the system depends most importantly on the expressiveness that will be required to faithfully model the key dynamics of the system. For instance, a reactive system with finite states without a notion of time can be modeled using Labeled Finite State Automata (LFSA). On the other hand a time-critical system cannot be modeled using the same LFSA. They will need a formalism that can either model time or temporal ordering among the states of the system. Timed Automata (TA) and Timed Petri-Nets (TPN) are two such formalisms that have been used to model time-critical systems. Though such formalisms can indeed model timing characteristics of time critical systems, they are not panacea. In fact, real-time cyber-physical systems that involve complex dynamics and stochastic behavior are not expressible by these formalisms [139]. Also, the model checking of such systems is undecidable and one thing that can be done is to approximate them with the available formalisms [140]. Alternatively, this problem can be solved by incorporating the formalism that can model the stochastic and non-linear dynamical behavior of the system and then exploiting the technique of Statistical Model Checking (SMC) [141]. The main idea behind SMC is to make an executable model of the system under consideration and perform a finite number of simulations. Results of these simulations are monitored and are used by the statistical techniques such as sequential hypothesis and Monte-Carlo simulations to find whether the system satisfies the required property with some given degree of confidence. One limitation of SMC is that it provides results not on the basis of exhaustive exploration, rather, a bounded number of simulations. Due to this fact, SMC is considered as a compromise between testing and the classical model checking. Though SMC is not as powerful as the classical model checking, it is still equivalent to running an exponential number of simulations [121].

Though TPN is capable of modeling real-time and time-sensitive systems, tools present for model checking TPN models are not sophisticated enough to support statistical model checking [142]. Timed Automata, on the other hand, has tools such as Uppaal and Prism that are sophisticated, well maintained and also allow statistical analysis of the model. Apart from

these, other modeling formalisms are Promela (of the SPIN model checker), CSP (of FDR model checker), TLA+ (of TLC model checker), Event-B, UML, Z, and so forth. A survey of existing tools for formal verification is given in Reference [143]. Out of all these model checkers, we have used the Uppaal model checker because of the following advantages it offers.

1. It works with models developed in Timed Automata (and their extensions) formalism.
2. It supports statistical model checking in Uppaal-SMC extension.
3. It supports the dynamic instantiation of templates
4. It supports graphical modeling which is an intuitive and easy way of modeling.
5. It offers high-level data structures and functions.
6. It is available for academic use without any cost, it is well maintained and has a big active community of users.

Out of these, features 2, 3, 4 and 5 are the ones that have been exploited in this work and their presence as a combination is the main reason for using this particular model checker.
In the presented work, we perform formal verification of the Heuristic Autonomous Intersection Management (HAIM) algorithm, which is presented in Chapter 5 of this thesis, using the Uppaal Model Checker, particularly its SMC variant. Verifying the HAIM algorithm for "No Collision" property will involve modeling traffic injection, vehicle behavior, Intersection Manager (IM), and collision detection procedure. As formal verification is aimed towards developing the system right, we perform verification at every stage of implementation of the HAIM algorithm. As we shall see, in the HAIM algorithm, we resolve conflicts in 4 stages where every following stage tries to resolve conflicts unresolved in the earlier stage. Verification is done after the implementation of each of these stages. This will guide the implementation of the HAIM algorithm and will also let us verify the claimed behaviors of each of these stages. After vehicles are scheduled by the HAIM algorithm, vehicles will travel with the assigned velocity and collision detectors will then check for collisions at every step of the simulation. Verification engine analyses these models for several runs against specified properties to check for their compliance. Furthermore, we perform implementation

verification by checking the satisfiability of some invariant conditions on the execution of the model and error injection to perform sanity checking of the model. In other words, we propose and demonstrate how to exploit the advantages of SMC in formal verification and alongside verifying sane modeling of the system.

7.1.3 Uppaal Model Checker

Uppaal [144] is a verification tool suite appropriate for verification of systems that can be modeled as a collection of non-deterministic processes with finite control structure and real-valued clocks communicating through channels or shared variables. Uppaal consists of three main parts—a description language, a simulator, and a model checker. The system to be verified is modeled as a group of TA. The description language is used to define variables, function definitions, and implement the inherent logic. A TA consists of various locations and these locations are connected by using edges. The edges are annotated with selections, guards, updates, and synchronizations. The purpose of these annotations is explained below.

selection: To non-deterministically bind a given identifier to a value in a given range.

update: Expression that is evaluated when the edge is traversed. In Uppaal, we can use functions in update expression.

guard: Expression that has to evaluate to true for the edge to be traversed.

synchronization: Processes communicate and synchronize using channels. This annotation may work as a publisher or a receiver on the edge.

Locations in Uppaal are labeled with invariants. Invariants are expressions that should always evaluate to true for the time the system is in that particular location. Locations in Uppaal can be made Urgent or Committed. These are the locations in which time is not allowed to pass that is, they freeze time. Furthermore, if a process is in a committed location, then the next transition must involve an edge from one of the committed locations.

Uppaal requirement specification supports five types of properties given in Table 7.1. In this table, p and q are state properties specified with atomic proposition; A and E are path quantifiers that stand for Always and Existential respectively and $[]$ and $<>$ are state quantifiers that stand for 'for all states' and 'for some state' respectively.

For systems that need dynamic creation and termination of automata and/or needs as an

Table 7.1: Properties in Uppaal.

Name	**Example**
Possibly	$E <> p$
Invariantly	$A[]\ p$
Potentially always	$E[]\ p$
Eventually	$A <> p$
Leads to	$p -- > q$

output the estimate of a probability, the standard Uppaal model checker will not suffice. For this reason, we choose Uppaal's Statistical Model Checker (SMC).

Uppaal-SMC [139] performs several runs of the system then uses results from statistics to get an overall estimate of the correctness of the system with respect to a property. Uppaal-SMC is different from the traditional Uppaal in the way that it allows us to specify the probability distribution that drives the timed behavior and also that the SMC engine offers the output of any query in probabilistic terms. The engine can: (i) Estimate the probability of an event, (ii) Compare the probability of an event with a value, and (iii) Compare two probabilities without computing them individually.

In Uppaal-SMC, to include the probabilistic nature, the query language is enhanced with the weighted version of the Metric Interval Temporal Logic (MITL) [145] which is defined by the following grammar.

$$\phi ::= p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathsf{O}\phi \mid \phi_1 \mathsf{U}^{x}_{\leq d} \phi_2,$$

where p is an atomic proposition, x is a clock, d is a natural number, O is the next operator, U is the Until operator and the logical symbols have their usual meanings. The weighted formula $\phi_1 \mathsf{U}^{x}_{\leq d} \phi_2$ is satisfied by the run if ϕ_2 is satisfied before the clock x exceeds d, and until that, ϕ_1 is satisfied. Two derived temporal operators known as time constrained

eventually ($\Diamond$), and time constrained *always* ($\Box$), are defined as follows.

$$\Diamond_x^{\leq d}\phi = true \ \mathsf{U}_d^x \ \phi \text{ and } \Box_x^{\leq d}\phi = \neg \ \Diamond_x^{\leq d}\neg\phi.$$

The eventually operator is defined in terms of the Until operator and the Always operator is defined in terms of the eventually operator. This allows us to write the following four types of queries in Uppaal-SMC.

1) *Quantitative analysis or probability estimation:* Estimates the probability of a path expression being true.
   For ex. $\boldsymbol{Pr}$ $[<= \boldsymbol{K}]$ $\{[]\ \boldsymbol{\phi}\}$, will give probability of ϕ to be true at every state when the system is run for K number of steps.

2) *Qualitative model checking or hypothesis testing:* Checks whether the probability of a property is less than or greater than the specified bound.
   For ex. $\boldsymbol{Pr}$ $[<= \boldsymbol{K}]$ $\{[]\ \boldsymbol{\phi}\} >= \boldsymbol{P}$, will state whether probability of ϕ to be true at every state when the system is run for K number of steps is greater than P or not.

3) *Probability comparison:* Compares two probabilities indirectly without estimating them.
   For ex. $\boldsymbol{Pr}$ $[<= \boldsymbol{K}]$ $\{[]\ \boldsymbol{\phi_1}\} <= \boldsymbol{Pr}$ $[<= \boldsymbol{K}]$ $\{[]\ \boldsymbol{\phi_2}\}$, will compare the two probabilities and return truth value of the expression.

4) *Value estimation:* Estimates the value of an expression by running a given number of simulations.
   For ex. $\boldsymbol{E}$ $[<= \boldsymbol{K}]$ $\{\boldsymbol{Var}\}$, will give the estimated value of the variable Var.

We next discuss the modeling of HAIM in Uppaal-SMC. We model our Intersection system using two automata, which are (i) Traffic Automaton and (ii) Master Automaton. The Master automaton is divided into three sections depending on the functionality, these three sections are:

- Vehicle Initialization Section,

- Controller Section, and

- Movement and Collision Check Section

We will next discuss these automata individually.

7.2 Modeling HAIM in Uppaal SMC

7.2.0.1 Traffic Automaton

Traffic automaton releases new vehicles into the system. The release of vehicles is done after a non-deterministic time interval given by the bound given in the guard expression of Figure 7.1. This bound is fixed to generate a traffic density of a minimum of 500 vehicles per hour and a maximum of 5000 vehicles per hour. This density range is the same as that used in Chapter 5. This spawning procedure that releases every vehicle after a non-deterministic time interval, mimics the traffic arrival pattern of vehicles in real-world traffic scenarios. With a simulation step of 0.1 s, *min_spawn_time* and *max_spawn_time* come out to be 7 (closest integer) and 72. To obtain these numbers we divide the total number of steps in one hour (which is 36000) by the number of vehicles to be spawned in one hour that is, 5000 and 500 respectively.

The Master automaton is declared as a dynamic template and it is instantiated every time a vehicle is spawned in the system. Dynamic templates are declared using the *spawn* function as can be seen in the Traffic automaton in Figure 7.1. To terminate a dynamic template, *exit* command is used. Also, there can be multiple instances of a dynamic template present at one time that means multiple master templates can exist at the same time with different levels of progression within their automaton. We will next discuss the *Master* template in detail.

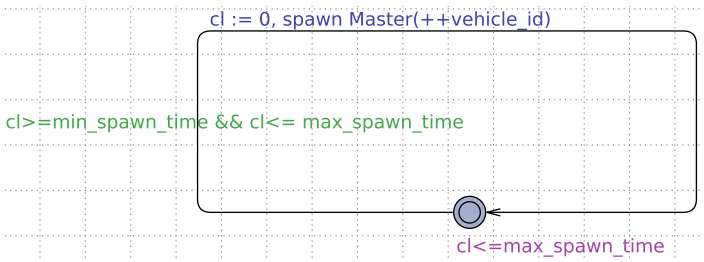


Figure 7.1: Traffic automaton.

7.2.0.2 Master Automaton

The Master automaton is as shown in Figure 7.2. The master automaton which is defined as a dynamic template, realizes three functionalities which are:

- Encoding the complete behavior of a vehicle starting from initialization, to traversing the approach region, the intersection region and the depart region.
- Encoding the entire processing that goes into calculating the target velocity for that vehicle using the HAIM algorithm.
- Encoding the collision detection procedure in approach region as well as intersection region.

Virtually we can divide the Master automaton into three sections which are: *Vehicle initialization section, Controller section, and Movement and collision check section.* Let us next discuss each of these sections in detail.

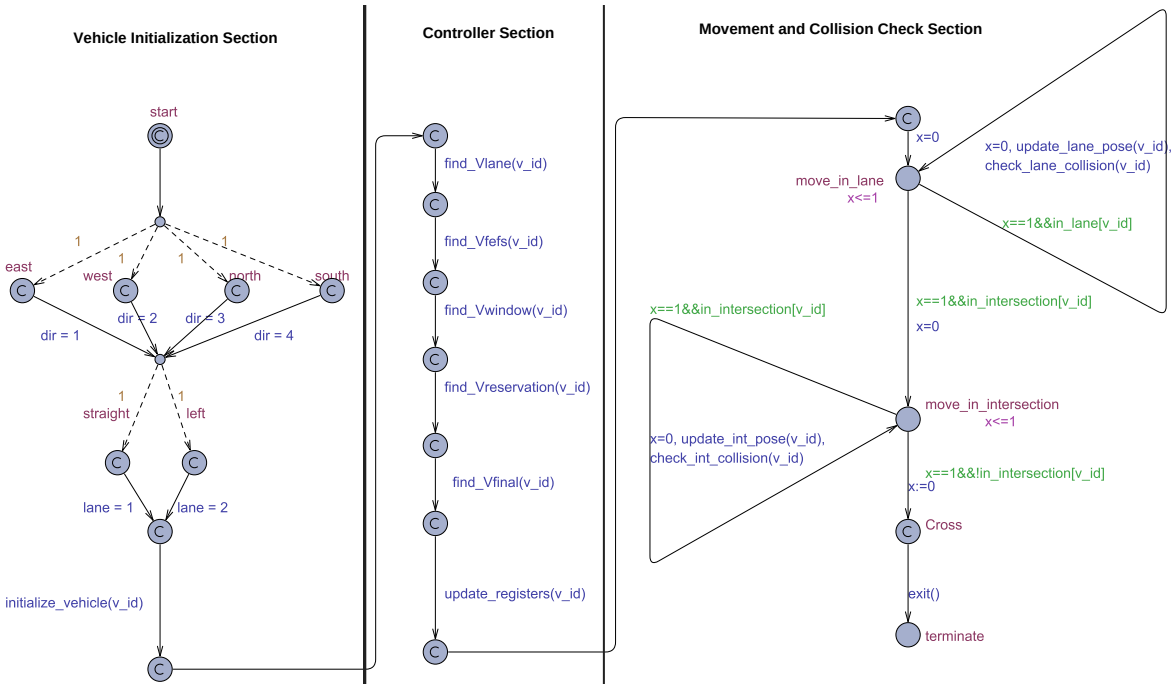


Figure 7.2: Master automaton.

7.2.0.3 Vehicle Initialization Section

The Master automaton starts in this section as it contains the initial location. Dashed edges following the initial location represent probabilistic edges with the relative weights specified as annotations. These dashed edges will randomly decide the source lane and the destination lane of the vehicle. All the weights are kept the same to get equally distributed traffic. The *initialize_vehicle()* function initializes the vehicle using the values of *dir* and *lane* variables. This function resolves the indexes of the conflict points that lie in the path of this vehicle and distances between successive conflict points are also assigned in this function. The arrival velocity of the vehicle is set randomly (less than maximum velocity).

7.2.0.4 Controller Section

This section represents the working of the Intersection Manager and implements the HAIM pipeline. The pipeline starts with calculating the lane velocity for the vehicle. Lane velocity is then used as the starting point for calculating the rest of the three velocities namely FEFS velocity, Window velocity, and the Reservation velocity. To decide which of these three velocities will be the final velocity, we implement the logic flow given in the flowchart in Figure 7.3, in the function *find_Vfinal()*.

It is this section of the Master automaton in which stage-wise implementation of HAIM algorithm is done. The $find\_Vfinal()$ function is attached to the outcomes of $find\_Vlane()$, $find\_Vfefs()$, $find\_Vwindow()$ and $find\_Vreservation()$ respectively in the corresponding implementation stage. We discuss more about this in the next section. The modeling of the HAIM algorithm in the Controller section of the Master automaton uses committed states to model the assumption that all of the communication and computation has been flawless and the vehicle has received its V_{final} that is, the velocity the vehicle has to attain after it leaves the buffer region. The movement of the vehicle starting from the end of the buffer region is modeled inside the Movement and Collision Check section of the Master automaton.

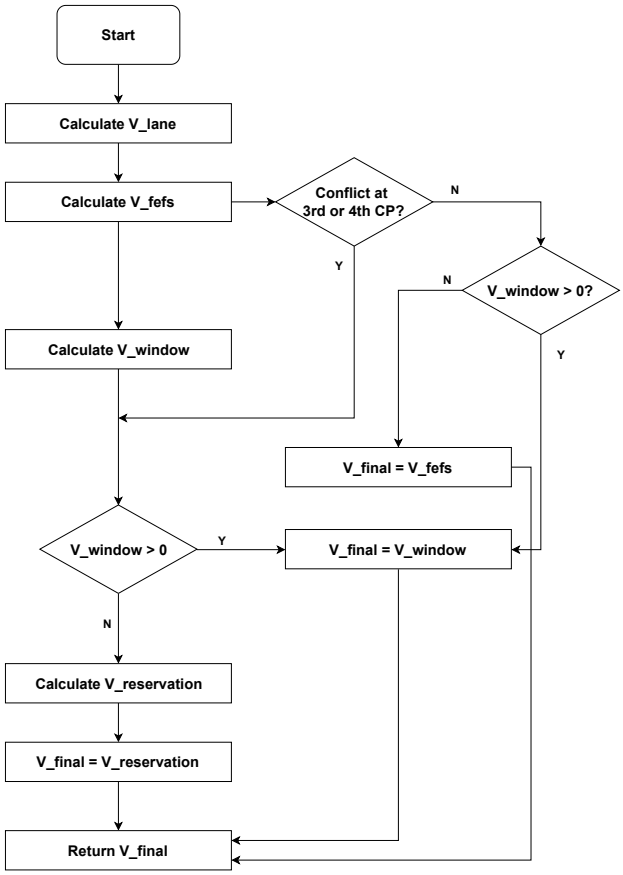


Figure 7.3: Decision flow of the HAIM algorithm

7.2.0.5 Movement and Collision Check Section

After leaving the buffer region, the vehicle will travel in the lane and first of all, make a transition from the entering velocity to the assigned velocity (V_{final}). At each step, the position and velocity of the vehicle are updated. For the time the vehicle is travelling in the lane, the Master automaton will be in the location *move_in_lane*. When the vehicle comes inside the intersection area, the automaton will come into the *move_in_intersection* location. Functions *update_lane_occupancy* and *update_int_occupancy* update the position of the vehicle in the lane and the intersection occupancy vectors respectively. These positions are used by the lane and intersection collision detection routines.

To detect lane collisions, we use the discrete position of the vehicle given by the lane occupancy vector of the corresponding lane. Lane occupancy vector is an array of cells. Those cells which coincide with the position of the vehicle have a value of 1. To determine the occupied cells for a vehicle, we first find the cell corresponding to the head position of

Table 7.2: Functions used in Uppaal model.

Serial no.	Function name	Description
1	*initialize_vehicle()*	Initialize variables such as enter_vel, cp_indices, cp_distances, etc.
2	*find_Vlane()*	Iterate to find safe lane velocity.
3	*find_Vfefs()*	Iterate to find velocity that satisfies critical CP at FEFS stage.
4	*find_Vwindow()*	Iterate to find velocity that find window between present reservations.
5	*find_Vreservation()*	Iterate to find velocity that satisfies critical CP at reservation stage.
6	*find_Vfinal()*	Implements the logic flow diagram shown in Figure 7.3.
7	*update_lane_occupancy()*	Update lane position of vehicle in the lane occupancy vector.
8	*update_int_occupancy()*	Update intersection position of vehicle in the intersection occupancy vector.
9	*check_lane_collision()*	Checks in each lane occupancy vector for length of continuous occupancy.
10	*check_int_collision()*	Checks for simultaneous occupancy of CP by two vehicles.

the vehicle and then set all the cells to 1 that comes under the length of the vehicle. For simplicity, we have used in the simulation, the same length (3 met.) for all the vehicles. The procedure *update_lane_occupancy()* implements this logic.

Now, to detect a collision, we will look for the number of cells for which there is continuous occupancy. If the length corresponding to continuous occupancy of cells crosses the length of one vehicle then that would mean there is a collision between two vehicles. Illustration is shown in Figure 7.4. Complete overlap of occupancy of two vehicles would go undetected. However, with the limits of velocity and the dimensions of vehicles used, it would not be possible for two occupancies to be non-overlapping at one step and completely overlapping at the very next step because that would require relative distance of at least the length of the vehicle (3 m) to be covered in one times step which is greater than maximum possible value (1.7 m). The collision condition is checked for each lane occupancy vector at every simulation step. For any positive detection, a lane collision counter is incremented. The function *check_lane_collision()* performs the above mentioned operation. The guard, invariant and update of clock variable x make sure that the collision is run at every step of the simulation. When the vehicle is through the intersection area, the Master template will be terminated.

The intersection collision detection routine works in the same fashion as that of lane collision detection. Here also, the position of the vehicle inside the intersection is represented using discrete cells inside the intersection occupancy vector. As we have modeled vehicle occupancy using line segment, the width of the vehicle is not modeled. This was not a con-

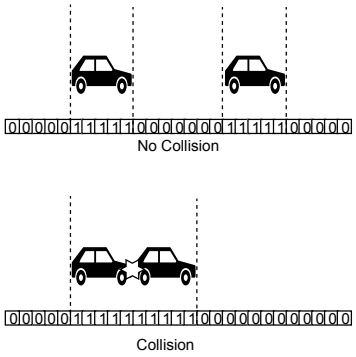
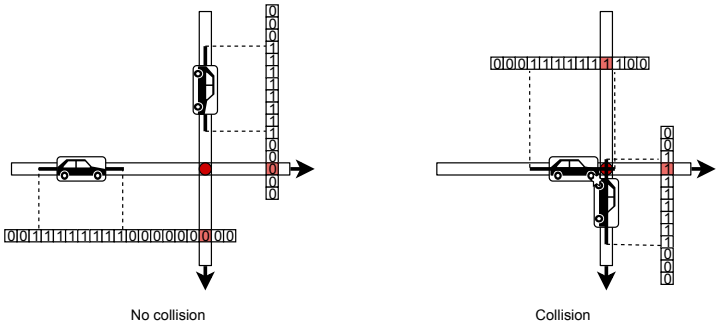


Figure 7.4: Lane collision detection.

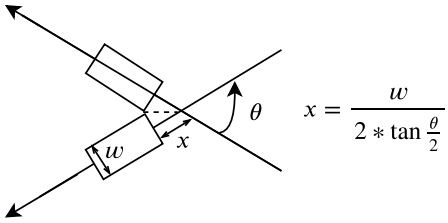
cern in the lane collision detection as vehicle always follow the order in which they entered. However, in case of collision detection in an intersection, the width has to be modeled. Along with width, we also need to model the orientation of vehicle in the intersection due to the angle of intersection at the conflict point. Angle of intersection is as depicted in Figure 7.5b. To model these two, we use extended occupancy of vehicles. This means, the line segment corresponding to the length of the vehicle is extended on both ends by a width given by the expression shown in Figure 7.5b, this figure also shows how this expression models the geometry of vehicle at the conflict point. The occupancy of cells of these vectors is updated at every step for each vehicle in the simulation. Like in lane collision detection, we calculate the position of the cell corresponding to the head of the vehicle and then find the trailing occupancy cells using the length of the vehicle. Implementation is done inside the function $update\_int\_occupancy()$.

To detect the intersection collision, we use a different approach than in the case of lane collisions. The cells corresponding to all the conflict points are determined in all the lane to lane connections. Width of vehicles is incorporated in collision detection by extending the intersection occupancy of vehicles on both sides by a length given in terms of the width of the vehicle in Figure 7.5b. This way we detect the intersection collision of vehicles using their occupancy only at common conflict point. Simultaneous occupancy by two vehicles of the cell corresponding to a common conflict point will mean that there is a collision in the intersection. If that happens, an intersection collision counter is incremented. The function *check_int_collision()* performs the above mentioned operation. The guard, invariant and update of clock variable *x* make sure that the collision check is performed at every step of the

simulation. Figure 7.5a illustrates the working of the intersection collision detection routine. Table 7.2 lists all the functions used in the Uppaal-SMC model of the HAIM algorithm.



(a) Intersection collision detection.



(b) Occupancy extension in terms of vehicle width and angle of intersection.

Figure 7.5: Intersection collision detection logic and occupancy extension to model vehicle width and angle of intersection

7.3 Verification

To perform the sanity check of the modeling of the HAIM algorithm in Uppaal-SMC, we perform verification to verify the claims made about the properties of the reservation made by the three constituent layers along with verifying the safety property for the complete heuristic. To do this, we model the HAIM algorithm in the following stages.

1) Model with only V_{lane}.

2) Model with V_{lane} and V_{fefs}.

3) Model with V_{lane}, V_{fefs} and V_{window}.

4) The complete model with all the velocities.

At each of these stages, queries are run to get the probability (confidence) of the model in resolving lane and intersection collisions. Since we put the collision count in counters called $lane\_collisions$ and $int\_collisions$, to determine the confidence of model in resolving collisions, we encode a query that has a literal meaning as: *What is the probability that the count of the given counter is always zero when the model is run for a bounded (given) number of steps*. In Uppaal-SMC, this query when translated for lane collision and intersection collisions both look as

$$Pr\ [<=\ K]\ \{[]\ lane\_collisions == 0\}, and$$
$$Pr\ [<=\ K]\ \{[]\ int\_collisions == 0\}$$

These are the queries that are passed to the model checker. Here, K is the number of steps for which the model is run. We perform verification for values of K ranging from 1000 to 5000, this way we can observe the evolution of the behavior of the model at each stage of implementation. Choosing a larger value of K is constrained by the time the model checking tool takes to return results. For $K = 5000$, the verifier took close to 88 days to give output for query corresponding to the complete model.

Verification results are tabulated in Table 7.3. This table contains verification results corresponding to five values of K which are 1000, 2000, 3000, 4000, and 5000. For each value of K, there is a verification result corresponding to the 4 implementation stages mentioned above. Verification results corresponding to no lane collision and no intersection collision are given in columns 3 and 4 of the table respectively. Columns 5 and 6 give results for artificial error injected systems as explained in Section 7.4.2.

7.3.1 Model with Lane Velocity As the Final Velocity

The lane velocity is calculated by making the depart time of the current vehicle greater than that of the previous vehicle in the same lane. This velocity is the maximum velocity with which the vehicle can travel that will not result in a collision in the lane. However, it can result in a collision inside the intersection. Thus the claim made about the model with lane velocity as the final velocity is that the vehicle will not undergo any lane collision but it can result in an intersection collision. Thus the statistical model checker should return the

Table 7.3: Verification results

Steps	**Velocities Used**	**Pr(no_lane_col)**	**Pr(no_int_col)**	**Pr(no_int_col) in Erroneous System (IM)**	**Pr(no_int_col) in Erroneous System (Veh)**
1000	V_{lane}	[0.99, 1]	[0.772123, 0.792123]	[0.757588, 0.857588]	[0.848374, 0.948374]
	V_{fefs}	[0.99, 1]	[0.971126, 0.991126]		
	V_{fefs} and V_{window}	[0.99, 1]	[0.982451, 1]		
	V_{fefs} , $V_{win.}$ & $V_{res.}$	[0.99, 1]	[0.99,1]		
2000	V_{lane}	[0.99, 1]	[0.567835, 0.587835]	[0.599051, 0.699051]	[0.721003, 0.821003]
	V_{fefs}	[0.99, 1]	[0.950781, 0.970781]		
	V_{fefs} and V_{window}	[0.99, 1]	[0.974259, 0.994259]		
	V_{fefs} , $V_{win.}$ & $V_{res.}$	[0.99, 1]	[0.99, 1]		
3000	V_{lane}	[0.99, 1]	[0.411410, 0.431410]	[0.424255, 0.524255]	[0.664092, 0.764092]
	V_{fefs}	[0.99, 1]	[0.930020, 0.950020]		
	V_{fefs} and V_{window}	[0.99, 1]	[0.965653, 0.985653]		
	V_{fefs} , $V_{win.}$ & $V_{res.}$	[0.99, 1]	[0.99, 1]		
4000	V_{lane}	[0.99, 1]	[0.296923, 0.316923]	[0.317209, 0.417209]	[0.563821, 0.663821]
	V_{fefs}	[0.99, 1]	[0.912165, 0.932615]		
	V_{fefs} and V_{window}	[0.99, 1]	[0.954518, 0.974518]		
	V_{fefs} , $V_{win.}$ & $V_{res.}$	[0.99, 1]	[0.99, 1]		
5000	V_{lane}	[0.99, 1]	[0.213275, 0.233275]	[0.263008, 0.363008]	[0.48794, 0.58794]
	V_{fefs}	[0.99, 1]	[0.89438, 0.914348]		
	V_{fefs} and V_{window}	[0.99, 1]	[0.947572, 0.967572]		
	V_{fefs} , $V_{win.}$ & $V_{res.}$	[0.99, 1]	[0.99, 1]		

maximum possible confidence value for no lane collision but not for no intersection collision. Verification results are given in Table 7.3 in rows corresponding to V_{lane}.

7.3.2 Model with FEFS Velocity As the Final Velocity

The FEFS velocity is calculated by resolving conflicts only for the first two conflict points. So, there is a possibility that a slow-moving vehicle that entered before the current vehicle has a reservation at the third or the fourth conflict point at a time which overlaps with the current vehicle's schedule. Such conflicts are not resolved in the FEFS scheme so when the FEFS scheme is used alone, it may result in intersection collisions. However, since the FEFS scheme starts from the lane velocity and goes in the decrement mode to find FEFS velocity, we can say that FEFS velocity is always less than or equal to the lane velocity. Hence, there should not be any lane collisions. Also, since the FEFS scheme resolves various intersection collisions, the probability of no intersection collision is expected to be more than that in the previous stage. Verification results are given in Table 7.3 in rows corresponding to V_{fefs}.

7.3.3 Model with FEFS Velocity and Window Velocity

Window scheme performs a window search operation within the previous reservations and its main aim is to further reduce the delay after the FEFS scheme by finding a greater velocity than the FEFS velocity if possible. It is claimed in the HAIM algorithm that the Window velocity is guaranteed to return a collision-free velocity if it returns a velocity. And in case it cannot find a window, the V_{fefs} will be the final velocity. So at this stage, the final velocity is given by window scheme if the V_{window} exists; otherwise, the V_{fefs} is assigned as the final velocity.

Since both FEFS and Window schemes return a velocity that is smaller than or equal to the Lane velocity, there should not be any lane collision caused in this stage as well. Also, as some of the conflicts unresolved by the FEFS scheme are resolved by the Window scheme, we expect a greater probability of no intersection collision than in the previous stage. Verification results are given in Table 7.3 in rows corresponding to V_{fefs} and V_{window}.

7.3.4 Complete Model with FEFS, Window and Reservation Velocities

Here we have all the constituent layers of HAIM, that means, this model should result in a collision-free trip through the lane and the intersection. The logic flow shown in Figure 7.3a is used to decide the final velocity. Verification results are given in Table 7.3 in rows corresponding to V_{fefs} , V_{window} & $V_{reservation}$. The verification times for this stage of implementation are given in the Table 7.4.

Table 7.4: Verification timings

S.No.	Number of Steps	Verification Time
1	500	8.5 hrs
2	1000	1.63 days
3	2000	7.8 days
4	3000	20.47 days
5	4000	42.2 days
6	5000	88 days

7.4 Implementation Verification

So far, we have introduced the HAIM algorithm, modeled it in the formalism of probabilistic timed automata using the Uppaal-SMC tool and presented verification results corresponding to four different stages of implementation. All this has been presented with an underlying assumption that the modeling of the HAIM algorithm is done faithfully. In fact, the majority of works in literature have this underlying assumption. This assumption leaves a possibility of false positive. In an attempt to prove the sanity of the modeling performed, correctness verification of the modeling itself is presented in this section.

7.4.1 Invariant Satisfiability

To prove that the model of HAIM correctly captures its properties, we define invariant conditions that the model should satisfy at every instant of the run. These properties are directly inherited from the HAIM algorithm itself. In other words, these properties represent invariant conditions over the execution of the algorithm and their fulfillment would mean faithful translation of the system. Following is the list of properties verified.

1. $0 \leq V_{fefs} \leq V_{lane}$**:**
   Satisfaction of this property will assert that V_{fefs} is always less than V_{lane} as claimed in the HAIM algorithm. The encoded CTL property is shown on line number 1 of Table 7.5

2. $0 \leq V_{window} \leq V_{lane}$
   Satisfaction of this property will assert that V_{window} is always less than V_{lane} as claimed in the HAIM algorithm.The encoded CTL property is shown on line number 2 of Table 7.5

3. $0 \leq V_{reservation} \leq V_{lane}$
   Satisfaction of this property will assert that $V_{reservation}$ is always less than V_{lane} as claimed in the HAIM algorithm.The encoded CTL property is shown on line number 3 of Table 7.5

4. $V_{reservation} \leq V_{fefs}$**:**

As $V_{reservation}$ considers all four CP instead of the only first two and as a result $V_{reservation}$ is the velocity that is minimum in all satisfying velocities of 4 conflict points. The encoded CTL property is shown on line number 4 of Table 7.5

5. $V_{reservation} \leq V_{window}$ (if V_{window} exists)**:**
   As V_{window} finds window between reservations below the top of the reservation record, V_{window} is not less than $V_{reservation}$ if V_{window} exists. The encoded CTL property is shown on line number 5 of Table 7.5

Properties queried in the model checker and their results are given in Table 7.5.

Table 7.5: Properties checked for invariant satisfiability

S.no	**Property**	**Confidence**
1	Pr([] [0, 1000] V_{fefs} >= 0 && V_{fefs} <V_{lane})	[0.99, 1]
2	Pr([] [0, 1000] V_{window} >= 0 && V_{window} <V_{lane})	[0.99, 1]
3	Pr([] [0, 1000] $V_{reservation}$ >= 0 && $V_{reservation}$ <V_{lane})	[0.99, 1]
4	Pr([] [0, 1000] $V_{reservation}$ <= V_{fefs})	[0.99, 1]
5	Pr([] [0, 1000] V_{window} != 0 –>V_{window} >$V_{reservation}$)	[0.99, 1]

Along with verifying the above specified properties using the model checker, we also perform a simulation to visualize these properties. For this purpose, we plot the difference of the V_{fefs}, V_{window} and $V_{reservation}$ from V_{lane} as shown in Figure 7.6. A non-negative value throughout this graph shows that V_{lane} is always greater than the rest of the three velocities. Also, the relative magnitudes can be compared as specified by above properties. We can also notice one thing from this graph and it is that V_{fefs} almost coincides with $V_{reservation}$ except only at a few time steps. This tells us that only at few occasions, V_{fefs} is different from $V_{reservation}$. This confirms our claim that most of the conflicts are resolved at first two conflict points.

Values of the probability outputs given in Table 7.3 shows that vehicles actually cross the lane and intersection regions and reach their destination. However, to prove it, we insert in the Master automata, a state called *Cross* before the Terminate state. We then ask the simulator that how many templates are currently in the scenario and how many templates

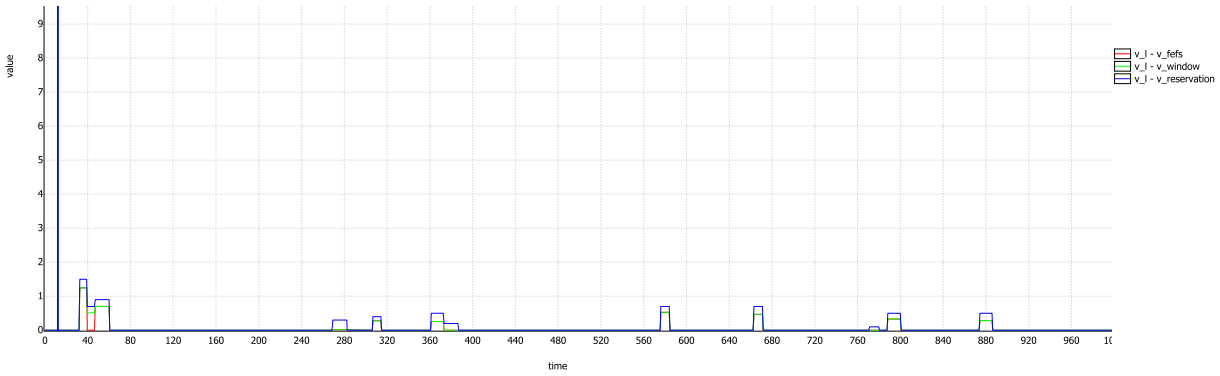


Figure 7.6: Difference of other velocities from V_{lane}.

has reached the *Cross* state. To get the output, the following query is passed to the verifier.

$$simulate\ 1\ [<= 500]\ numOf(Master),\ sum(v : Master)\ (v.Cross)$$

The graph that this query generates will depict the number of *Master* templates present at any given time in the simulation and also will give a spike of truth value of 1.0 every time a master template reaches the *Cross* state. The graph is shown in Figure 7.7. We can observe in this graph each spike of the truth value *v.Cross* corresponds to a decrease in the graph of *numOf(Master)*. This shows that *Master* templates actually traverse through the lane and intersection and get terminated.

7.4.2 Artificial Error Injection Testing

We will now deliberately inject errors into our model and see whether the verifier reflects the error in the model or not. If the error is reflected, it should mean that the model has deviated from a setting that was indeed correct. One could argue that if a system is incorrect, then introducing some error will make it more incorrect and hence, the error injection result should not prove anything. However, in the context of the presented subsection, where verification results have given affirmation of the correctness of the algorithm and also algorithm's behavior is proved to be preserved by the model as established by invariant satisfiability, error injection will indeed strengthen the confidence in overall implementation. To inject errors, we choose Vehicle and Controller (IM) as these are the two prominent entities. To inject an error in the IM, we choose the maximum satisfying velocity of all 4 CPs instead of the mini-

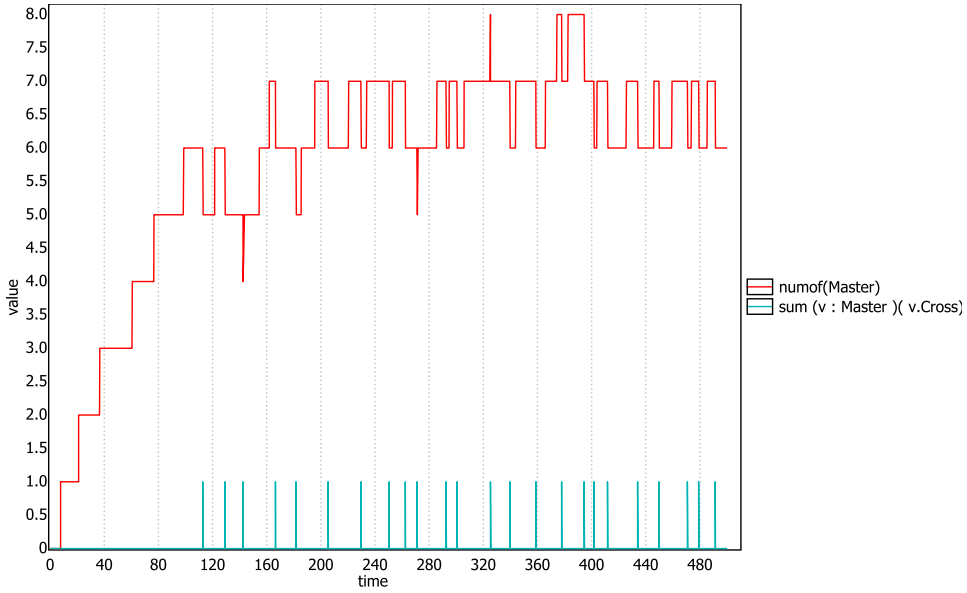


Figure 7.7: Number of active templates at any given time and templates getting terminated.

mum. This way the IM will resolve conflict only at one CP whereas, on other CPs, conflicts may occur. The results obtained confirm that the error injected system is making overlapping reservations which result in collisions as evident by the low probability in the output given in the second last column of Table 7.3.

To model vehicle malfunctioning, we make some (every tenth) of the vehicles in the simulation disobedient. These vehicles will just travel with their initial velocity disregarding other vehicles and the intersection manager. Outputs corresponding to this error injection is given in the last column of Table 7.3.

7.5 Discussion

Results given in Table 7.3, along with implying the correctness of the HAIM algorithm, also verify the claimed behaviors of its constituent four heuristics. Starting with the first stage of implementation which corresponds to V_{lane} only, we obtain maximum confidence for no lane collision and non-maximum confidence for no intersection collision, which is what we expected. As all other velocities are less than or equal to V_{lane}, we would expect that all of them result in maximum confidence in no lane collision. In the second stage of implementation, we use V_{fefs} only. As V_{fefs} resolves some of the intersection conflicts

but not all, we would expect the confidence for no intersection collision at this stage to be not maximum but more than that in the case of V_{lane}. Similarly, as V_{window} resolves some of the conflicts unresolved by V_{fefs} but not all, we expect at this stage greater confidence for no intersection collision but not maximum. The final stage shall resolve all the conflicts. Hence, it should result in maximum confidence for no lane collision as well as no intersection collision. As we can see in Table 7.3, all these expectations are met in each set of experiments corresponding to different values of K. Though the numbers presented in Table 7.3 does not contain much significance when considered alone. However, the evolution of these results with an increasing number of steps and with an increasing level of implementation of the algorithm, tells us about the behavior of the stages of implementation.

The methodology adopted to represent vehicles and their occupancy inside lane and the intersection has a small memory footprint, results in very efficient collision detection, and captures essential dynamics even in the limited resources of the verification tool. The geometry of the vehicle while crossing the conflict point is considered using the angle of intersection and extended occupancy. By performing the verification of the model, we demonstrate how internal verification can be done and argue about its importance in the complete verification. Apart from satisfying invariants, we perform two error injections in the system, one in the IM and another in the vehicle. However, the first error injection in which the IM is made to work in an erroneous manner should result in more number of collisions. This is because the IM will make incorrect reservations for all the incoming vehicles. Whereas, in the second error injection, only some of the vehicles are made disobedient. Due to this reason, we expect the probability of no intersection collision in the first erroneous system to be less than the second one. Meeting of all these expectations added with the proof of termination of Master templates, gives us the evidence of faithful modeling of the overall system. In fact, the stage-wise implementation and verification has helped us a lot in surfacing errors in implementation numerous times during the development of the HAIM model.

The query we give to the verifier translates to, "*What is the probability that there is no lane/intersection collision at all when the system is simulated for K number of steps*". Since V_{lane}, V_{fefs} and V_{window} do not guarantee the no-intersection-collision condition, the probability for no-intersection-collision for the first three stages of implementation does not show maximum confidence. Also, this probability should decrease with an increase in the

number of steps. Same is the case with error injected systems as well. For a simulation with a sufficient number of steps, these values should reach the minimum confidence range of the verifier however, the choice of simulation steps is restricted by the verification time. Though the work presented considers verification for 5000 steps, it demonstrates how we can model an intersection scenario, vehicle behavior, and collision detection and verify with limited resources of a model checker.

Verification results obtained for the model corresponding to the complete model with all four heuristic-levels give absolute confidence in resolving all conflicts and giving a collision free intersection control implementation. The confidence shown by the verifier for the complete heuristic is [0.99, 1] which represents the absolute confidence by the verifier. This output shows that the actual probability of no collision in the scenario can be anything between 0.99 and 1.0, both values included. This absolute confidence is shown only when in all the runs, verifier could not find any counter example and that is the reason it includes a probability of 1.0 in its result. Had it been the case that it found even a single counter-example, we would not have received this output. In SMC, the required precision can be adjusted to trade off with the number of runs verifier performs to give results which is a direct measure of the verification time the verifier takes.

Due to restriction on the number of steps for which verification is done because of the timing requirements, the present verification scheme is carried out for up to 5000 steps of 0.1 seconds each. Although, it is the first step, we believe it's a step in the right direction. The verification time observed in our procedure reflects the observations made in [138], which states that the timing requirements for testing an autonomous vehicle technology could take impractical amount of time. However, further research dedicated to decrease the verification time using parallel computations, state-space reduction, etc. would result in multi-fold reduction in it and would make verification play a critical role in generating correctness proofs for autonomous vehicles, related hardware and software modules and associated algorithms in the coming future.

7.6 Summary

In this chapter, we performed the correctness verification of the proposed HAIM algorithm against the "no collision" property. For this purpose, we used the Uppaal-SMC model checker which accepts input-system modeled in a formalism similar to the probabilistic time automata. Using quantitative model checking that gives results in probabilistic terms, along with verifying the "no collision" property, the claimed behaviors of the four heuristic levels of the HAIM algorithm are verified by applying statistical model checking at every stage of HAIM implementation. Later, to verify faithful modeling of the system, we performed implementation verification using invariant satisfiability, artificial error injection, and simulation based verification. Results obtained give evidence of the "no collision" property and faithful modeling with absolute confidence.

Chapter 8

Conclusion and Future Research Directions

8.1 Conclusion

In this dissertation, we presented our contributions with the objective of utilizing the capabilities of CAVs for enhancing the traffic management strategies in the form of two algorithms, one each for (i) cooperative lane changes of a group of vehicles and (ii) intersection management of autonomous vehicles. Before that, we introduced a new ITS procedure called Lane Sorting and discussed how it stands out of the existing ITS procedures of cooperative lane changing and platooning. The lane sorting procedure is introduced with an intention to accurately define objectives and requirements for scenarios that involve simultaneous lane changes of a group of vehicles within the available length of road. Defining these requirements and objectives accurately shall result in dedicated efforts that cater to such scenarios.

The proposed algorithm for lane sorting called CLS implements a cooperative algorithm in which incoming traffic is divided into independent batches called frames. To sort vehicles in a frame, a non-linear programming problem reducible to MILP was formed and then solved for optimum vehicle positions in the frame. The independence of every frame along with ensuring safety also offers freedom in terms of communication and computation model used and also in terms of redesigning of the cooperative lane sorting procedure for different applications for instance the CLS algorithm could also be used for cooperative lane

change of individual vehicles at highways. Simulations were performed for various traffic densities to obtain the average length of road required to sort all vehicles with different values of adjustable parameters (V_{common}, and *frameLen*). Repeated simulations results can be recorded for any given scenario to create a look-up table that can be referred to set the values of parameters such that vehicles are sorted within the available length.

The next contribution of this dissertation is the algorithm for intersection management of autonomous vehicles. We propose the Heuristic Autonomous Intersection Management (HAIM) algorithm which is a four-leveled heuristic algorithm that resolves all the conflicts in lane and intersection progressively. The HAIM algorithm demonstrates an application of the previously defined CLS algorithm by considering sorted traffic with respect to the destination direction at the intersection. The lane discipline along with pre-defined trajectories at intersections greatly reduced the complexity of the problem. Out of the four heuristics, the first one resolves all the rear-end collisions that a vehicle can have in lane whereas the later three heuristics progressively resolved all intersection conflicts. A comparative analysis for the HAIM algorithm was performed with two existing autonomous intersection management algorithms and also with traffic light control. Simulation results showed that the HAIM algorithm results in a minimum trip delay due to scheduling out of all. Also, the HAIM being a heuristic algorithm does not involve any computationally intensive procedure and has a complexity of $\mathcal{O}(n)$ with n being the number of vehicles incoming to the intersection.

The CLS and HAIM algorithm, both make use of a constant velocity traversal of vehicles. Having minimum changes in velocity offers benefits such as maximum comfort, minimum fuel consumption, maximum economy, minimum exhaust, etc. Also having a common velocity for all vehicles in a scenario allows us to keep a lower value of inter-vehicle distances such as in platooning and also in the CLS algorithm.

The next major contribution of this dissertation and the final one is the formal verification of the proposed HAIM algorithm to guarantee that all the lane and intersection conflicts are indeed resolved in an intersection controlled by the HAIM algorithm. We make use of the statistical model checking technique for this task. Along with verifying the no-collision property, techniques are employed to verify that faithful modeling of the scenario, vehicle behavior, HAIM algorithm, and collision detection were performed. For this purpose, a systematic combination of techniques such as internal verification using invariant satisfiabil-

ity, artificial error injection, and simulation-based verification was used. Verification results obtained showed absolute confidence in no collision property and faithful modeling of the overall system. The employed verification procedure also gave evidence of satisfaction of the claimed behaviors of the HAIM algorithm.

8.2 Future Research Directions

It is critical to put dedicated efforts towards exploration and implementation on how timing requirements of verification of autonomous vehicles and associated algorithms can be reduced for instance, by parallel computation. We would pursue this line of research in the future.

Enhancing the efficiency and safety of lane changing operations by utilizing cooperative property of CAVs could make a huge difference in traffic management systems as lane changing is one of the most common maneuvers in driving. Extending the proposed CLS to dedicated scenarios such as highways, urban roads, curved roads, etc. and investigating the improvements it results could be a possible advancements of the presented work.

The proposed HAIM algorithm has shown promising results and extending it to a traffic that schedules a traffic involving both autonomous vehicles and manually driven vehicle would be an extension that will find relevance in the near future.

Along the similar line as verification of the HAIM algorithm, we would like to pursue the verifciation of the CLS algorithm to ensure the property of collision freedom. We would also like to enumerate various scenarios and their respective maneuvering algorithms, especially those in which there is a chance for collision. Then verify such algorithms to evaluate the safety property of "collision freedom".

Bibliography

[1] "History of the automobile." https://www.britannica.com/technology/automobile/History-of-the-automobile. Accessed: 2020-06-14.

[2] N. Mccarthy, "Traffic congestion costs u.s. cities billions of dollars every year." https://www.forbes.com/sites/niallmccarthy/2020/03/10/traffic-congestion-costs-us-cities-billions-of-dollars-every-year-infographic/#620087674ff8. Accessed: 2020-06-14.

[3] Businesstoday.in, "India wastes rs 1.44 lakh crore due to traffic congestion, says uber study." https://www.businesstoday.in/current/economy-politics/india-wastes-as-much-as-rs-1-44-lakh-crore-due-to-traffic-congestion-says-uber-study/story/275194.html. Accessed: 2020-06-14.

[4] U.S.E.P.A, "Greenhouse gas emissions from a typical passenger vehicle." https://www.epa.gov/greenvehicles/greenhouse-gas-emissions-typical-passenger-vehicle. Accessed: 2020-06-14.

[5] M. Papageorgiou, C. Diakaki, V. Dinopoulou, A. Kotsialos, and Y. Wang, "Review of road traffic control strategies," *Proceedings of the IEEE*, vol. 91, no. 12, pp. 2043–2067, 2003.

[6] W.H.O., "The top 10 causes of death." https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death. Accessed: 2020-06-14.

[7] N. H. T. S. Administration *et al.*, "2016 fatal motor vehicle crashes: overview," *Traffic Safety Facts Research Note*, vol. 2017, p. 10, 2017.

[8] R. Horowitz and P. Varaiya, "Control design of an automated highway system," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 913–925, 2000.

[9] DARPA, "The grand challenge." https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles. Accessed: 2020-06-14.

[10] E. Dijkstra, "Testing shows the presence, not the absence of bugs," in *Software Engineering Techniques: Report of a conference sponsored by the NATO Science Committee (Robert M. McClure, 2001 ed.). NATO, Scientific Affairs Division, Brussels, Rome, Italy*, p. 16, 1969.

[11] B. Alpern and F. B. Schneider, "Recognizing safety and liveness," *Distributed computing*, vol. 2, no. 3, pp. 117–126, 1987.

[12] S. Carty and J. Healey, "Toyota recalls millions more; another accelerator issue hits carmaker," *USA Today.[January 22]*, 2010.

[13] I. Corp., "Annual report-1994." `https://www.intel.com/content/www/us/en/history/history-1994-annual-report.html`. Accessed: 2020-09-19.

[14] N. H. T. S. Administration *et al.*, "Us department of transportation releasesâĂŹ preparing for the future of transportation: Automated vehicles 3.0âĂŹ," *US Department of Transportation*, 2018.

[15] N. H. T. S. Administration *et al.*, "Automated driving systems 2.0: A vision for safety," *Washington, DC: US Department of Transportation, DOT HS*, vol. 812, p. 442, 2017.

[16] D. Bevly, X. Cao, M. Gordon, G. Ozbilgin, D. Kari, B. Nelson, J. Woodruff, M. Barth, C. Murray, A. Kurt, *et al.*, "Lane change and merge maneuvers for connected and automated vehicles: A survey," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 105–120, 2016.

[17] P. Kavathekar and Y. Chen, "Vehicle platooning: A brief survey and categorization," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 54808, pp. 829–845, 2011.

[18] R. Hall and C. Chin, "Vehicle sorting for platoon formation: Impacts on highway entry and throughput," *Transportation Research Part C: Emerging Technologies*, vol. 13, no. 5-6, pp. 405–420, 2005.

[19] Y. Luo, G. Yang, M. Xu, Z. Qin, and K. Li, "Cooperative lane-change maneuver for multiple automated vehicles on a highway," *Automotive Innovation*, vol. 2, no. 3, pp. 157–168, 2019.

[20] B. Li, Y. Zhang, Y. Ge, Z. Shao, and P. Li, "Optimal control-based online motion planning for cooperative lane changes of connected and automated vehicles," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3689–3694, IEEE, 2017.

[21] D. Desiraju, T. Chantem, and K. Heaslip, "Minimizing the disruption of traffic flow of automated vehicles during lane changes," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 3, pp. 1249–1258, 2014.

[22] N. H. T. S. Administration *et al.*, "Traffic safety facts: 2007," *Annals of Emergency Medicine*, vol. 53, no. 6, p. 824, 2009.

[23] J. M. Wing, "A specifier's introduction to formal methods," *Computer*, vol. 23, no. 9, pp. 8–22, 1990.

[24] J. Wang, *Formal Methods in Computer Science*. CRC Press, 2019.

[25] G. J. Holzmann, "The model checker spin," *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[26] C. Hoare, "A theory of csp," *Commun. ACM*, vol. 21, no. 8, 1978.

[27] L. Lamport, "Specifying concurrent systems with tlaˆ+," *NATO ASI SERIES F COMPUTER AND SYSTEMS SCIENCES*, vol. 173, pp. 183–250, 1999.

[28] J. Y. Halpern and M. Y. Vardi, "Model checking vs. theorem proving: a manifesto," *Artificial intelligence and mathematical theory of computation*, vol. 212, pp. 151–176, 1991.

[29] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pp. 46–57, IEEE, 1977.

[30] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Workshop on Logic of Programs*, pp. 52–71, Springer, 1981.

[31] J. E. Friedl, *Mastering regular expressions*. " O'Reilly Media, Inc.", 2006.

[32] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.

[33] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.

[34] A. Legay, B. Delahaye, and S. Bensalem, "Statistical model checking: An overview," in *International conference on runtime verification*, pp. 122–135, Springer, 2010.

[35] T.-S. Dao, C. M. Clark, and J. P. Huissoon, "Distributed platoon assignment and lane selection for traffic flow optimization," in *2008 IEEE Intelligent Vehicles Symposium*, pp. 739–744, IEEE, 2008.

[36] T.-S. Dao, C. M. Clark, and J. P. Huissoon, "Optimized lane assignment using inter-vehicle communication," in *2007 IEEE Intelligent Vehicles Symposium*, pp. 1217–1222, IEEE, 2007.

[37] A. Geiger, M. Lauer, F. Moosmann, B. Ranft, H. Rapp, C. Stiller, and J. Ziegler, "Team annieway's entry to the 2011 grand cooperative driving challenge," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 1008–1017, 2012.

[38] T. Robinson, E. Chan, and E. Coelingh, "Operating platoons on public motorways: An introduction to the sartre platooning programme," in *17th world congress on intelligent transport systems*, vol. 1, p. 12, 2010.

[39] S. Tsugawa and S. Kato, "Energy its: another application of vehicular communications," *IEEE Communications Magazine*, vol. 48, no. 11, pp. 120–126, 2010.

[40] B. Németh, A. Csikós, I. Varga, and P. Gáspár, "Road inclinations and emissions in platoon control via multi-criteria optimization," in *2012 20th Mediterranean Conference on Control & Automation (MED)*, pp. 1524–1529, IEEE, 2012.

[41] J. Zhang and P. A. Ioannou, "Longitudinal control of heavy trucks in mixed traffic: Environmental and fuel economy considerations," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 1, pp. 92–104, 2006.

[42] Y. Zhang and G. Cao, "V-pada: Vehicle-platoon-aware data access in vanets," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 5, pp. 2326–2339, 2011.

[43] A. Uchikawa, R. Hatori, T. Kuroki, and H. Shigeno, "Filter multicast: a dynamic platooning management method," in *2010 7th IEEE Consumer Communications and Networking Conference*, pp. 1–5, IEEE, 2010.

[44] M. Segata, B. Bloessl, S. Joerer, F. Dressler, and R. L. Cigno, "Supporting platooning maneuvers through ivc: An initial protocol analysis for the join maneuver," in *2014 11th Annual conference on wireless on-demand network systems and services (WONS)*, pp. 130–137, IEEE, 2014.

[45] L. Xiao and F. Gao, "Practical string stability of platoon of adaptive cruise control vehicles," *IEEE Transactions on intelligent transportation systems*, vol. 12, no. 4, pp. 1184–1194, 2011.

[46] A. Kesting and M. Treiber, "How reaction time, update time, and adaptation time influence the stability of traffic flow," *Computer-Aided Civil and Infrastructure Engineering*, vol. 23, no. 2, pp. 125–137, 2008.

[47] J. Zhao, M. Oya, and A. El Kamel, "A safety spacing policy and its impact on highway traffic flow," in *2009 IEEE Intelligent Vehicles Symposium*, pp. 960–965, IEEE, 2009.

[48] Y. Zhai, L. Li, G. R. Widmann, and Y. Chen, "Design of switching strategy for adaptive cruise control under string stability constraints," in *Proceedings of the 2011 American Control Conference*, pp. 3344–3349, IEEE, 2011.

[49] M. Atagoziyev, K. W. Schmidt, and E. G. Schmidt, "Lane change scheduling for autonomous vehicles," *IFAC-PapersOnLine*, vol. 49, no. 3, pp. 61–66, 2016.

[50] S. Samiee, S. Azadi, R. Kazemi, A. Eichberger, B. Rogic, and M. Semmer, "Performance evaluation of a novel vehicle collision avoidance lane change algorithm," in *Advanced Microsystems for Automotive Applications 2015*, pp. 103–116, Springer, 2016.

[51] B. Li, Y. Zhang, Y. Zhang, and N. Jia, "Cooperative lane change motion planning of connected and automated vehicles: A stepwise computational framework," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 334–338, IEEE, 2018.

[52] Y. Shi, Q. He, and Z. Huang, "Capacity analysis and cooperative lane changing for connected and automated vehicles: Entropy-based assessment method," *Transportation research record*, vol. 2673, no. 8, pp. 485–498, 2019.

[53] J. Nie, J. Zhang, W. Ding, X. Wan, X. Chen, and B. Ran, "Decentralized cooperative lane-changing decision-making for connected autonomous vehicles," *IEEE Access*, vol. 4, pp. 9413–9420, 2016.

[54] J. Hu, L. Kong, W. Shu, and M.-Y. Wu, "Scheduling of connected autonomous vehicles on highway lanes," in *2012 IEEE Global Communications Conference (GLOBECOM)*, pp. 5556–5561, IEEE, 2012.

[55] U. Khan, P. Basaras, L. Schmidt-Thieme, A. Nanopoulos, and D. Katsaros, "Analyzing cooperative lane change models for connected vehicles," in *2014 International Conference on Connected Vehicles and Expo (ICCVE)*, pp. 565–570, IEEE, 2014.

[56] A. Kesting, M. Treiber, and D. Helbing, "General lane-changing model mobil for car-following models," *Transportation Research Record*, vol. 1999, no. 1, pp. 86–94, 2007.

[57] J. Erdmann, "Lane-changing model in sumo," *Proceedings of the SUMO2014 modeling mobility with open data*, vol. 24, pp. 77–88, 2014.

[58] B. Li, Y. Zhang, Y. Feng, Y. Zhang, Y. Ge, and Z. Shao, "Balancing computation speed and quality: A decentralized motion planning method for cooperative lane changes of connected and automated vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 3, pp. 340–350, 2018.

[59] Y. Luo, Y. Xiang, K. Cao, and K. Li, "A dynamic automated lane change maneuver based on vehicle-to-vehicle communication," *Transportation Research Part C: Emerging Technologies*, vol. 62, pp. 87–102, 2016.

[60] X. Fu, Y. Jiang, D. Huang, K. Huang, and J. Wang, "Trajectory planning for automated driving based on ordinal optimization," *Tsinghua Science and Technology*, vol. 22, no. 01, pp. 62–72, 2017.

[61] J. Suh, B. Kim, and K. Yi, "Stochastic predictive control based motion planning for lane change decision using a vehicle traffic simulator," in *2016 IEEE Transportation Electrification Conference and Expo, Asia-Pacific (ITEC Asia-Pacific)*, pp. 900–907, IEEE, 2016.

[62] K. Dresner and P. Stone, "Multiagent traffic management: A reservation-based intersection control mechanism," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 530–537, 2004.

[63] K. Dresner and P. Stone, "A multiagent approach to autonomous intersection management," *Journal of artificial intelligence research*, vol. 31, pp. 591–656, 2008.

[64] T.-C. Au and P. Stone, "Motion planning algorithms for autonomous intersection management," in *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[65] J. D. Little, "A proof for the queuing formula: L= λ w," *Operations research*, vol. 9, no. 3, pp. 383–387, 1961.

[66] T.-C. Au, N. Shahidi, and P. Stone, "Enforcing liveness in autonomous traffic management," in *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[67] D. Carlino, S. D. Boyles, and P. Stone, "Auction-based autonomous intersection management," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pp. 529–534, IEEE, 2013.

[68] G. Sharon and P. Stone, "A protocol for mixed autonomous and human-operated vehicles at intersections," in *International Conference on Autonomous Agents and Multiagent Systems*, pp. 151–167, Springer, 2017.

[69] M. Vasirani and S. Ossowski, "A computational market for distributed control of urban road traffic systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 2, pp. 313–321, 2011.

[70] L. Walras, *Éléments d'économie politique pure, ou, Théorie de la richesse sociale*. F. Rouge, 1896.

[71] M. Vasirani and S. Ossowski, "A market-inspired approach for intersection management in urban road traffic networks," *Journal of Artificial Intelligence Research*, vol. 43, pp. 621–659, 2012.

[72] I. H. Zohdy and H. Rakha, "Game theory algorithm for intersection-based cooperative adaptive cruise control (cacc) systems," in *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pp. 1097–1102, IEEE, 2012.

[73] Q. Jin, G. Wu, K. Boriboonsomsin, and M. Barth, "Advanced intersection management for connected vehicles using a multi-agent systems approach," in *2012 IEEE Intelligent Vehicles Symposium*, pp. 932–937, IEEE, 2012.

[74] C. Yu, G. Tan, and Y. Yu, "Make driver agent more reserved: An aim-based incremental data synchronization policy," in *2013 IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks*, pp. 198–205, IEEE, 2013.

[75] X. Wei, G. Tan, and N. Ding, "Batch-light: An adaptive intelligent intersection control policy for autonomous vehicles," in *2014 IEEE International Conference on Progress in Informatics and Computing*, pp. 98–103, IEEE, 2014.

[76] M. M. Abdelhameed, M. Abdelaziz, S. Hammad, and O. M. Shehata, "A hybrid fuzzy-genetic controller for a multi-agent intersection control system," in *2014 international conference on engineering and technology (ICET)*, pp. 1–6, IEEE, 2014.

[77] K. Zhang, A. De La Fortelle, D. Zhang, and X. Wu, "Analysis and modeled design of one state-driven autonomous passing-through algorithm for driverless vehicles at intersections," in *2013 IEEE 16th International Conference on Computational Science and Engineering*, pp. 751–757, IEEE, 2013.

[78] K. Zhang, A. Yang, H. Su, A. de La Fortelle, K. Miao, and Y. Yao, "Service-oriented cooperation models and mechanisms for heterogeneous driverless vehicles at continuous static critical sections," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 7, pp. 1867–1881, 2016.

[79] Y. Chang and P. Edara, "Arebic: Autonomous reservation-based intersection control for emergency evacuation," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1887–1892, IEEE, 2017.

[80] F. Yan, M. Dridi, and A. El Moudni, "Autonomous vehicle sequencing algorithm at isolated intersections," in *2009 12th International IEEE conference on intelligent transportation systems*, pp. 1–6, IEEE, 2009.

[81] J. Wu, A. Abbas-Turki, and A. El Moudni, "Cooperative driving: an ant colony system for autonomous intersection management," *Applied Intelligence*, vol. 37, no. 2, pp. 207–222, 2012.

[82] F. Perronnet, A. Abbas-Turki, and A. El Moudni, "A sequenced-based protocol to manage autonomous vehicles at isolated intersections," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pp. 1811–1816, IEEE, 2013.

[83] X.-F. Xie and Z.-J. Wang, "Siv-dss: Smart in-vehicle decision support system for driving at signalized intersections with v2i communication," *Transportation Research Part C: Emerging Technologies*, vol. 90, pp. 181–197, 2018.

[84] J. Lee and B. Park, "Development and evaluation of a cooperative vehicle intersection control algorithm under the connected vehicles environment," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 1, pp. 81–90, 2012.

[85] M. A. S. Kamal, J.-i. Imura, A. Ohata, T. Hayakawa, and K. Aihara, "Coordination of automated vehicles at a traffic-lightless intersection," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pp. 922–927, IEEE, 2013.

[86] K.-D. Kim, "Collision free autonomous ground traffic: A model predictive control approach," in *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, pp. 51–60, 2013.

[87] C. Wuthishuwong and A. Traechtler, "Vehicle to infrastructure based safe trajectory planning for autonomous intersection management," in *2013 13th international conference on ITS telecommunications (ITST)*, pp. 175–180, IEEE, 2013.

[88] N. Murgovski, G. R. de Campos, and J. Sjöberg, "Convex modeling of conflict resolution at traffic intersections.," in *CDC*, pp. 4708–4713, 2015.

[89] F. Altché, X. Qian, and A. de La Fortelle, "Least restrictive and minimally deviating supervisor for safe semi-autonomous driving at an intersection: An miqp approach," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2520–2526, IEEE, 2016.

[90] E. R. Müller, R. C. Carlson, and W. K. Junior, "Intersection control for automated vehicles with milp," *IFAC-PapersOnLine*, vol. 49, no. 3, pp. 37–42, 2016.

[91] S. A. Fayazi, A. Vahidi, and A. Luckow, "Optimal scheduling of autonomous vehicle arrivals at intelligent intersections via milp," in *2017 American control conference (ACC)*, pp. 4920–4925, IEEE, 2017.

[92] M. Bashiri and C. H. Fleming, "A platoon-based intersection management system for autonomous vehicles," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 667–672, IEEE, 2017.

[93] M. Bashiri, H. Jafarzadeh, and C. H. Fleming, "Paim: Platoon-based autonomous intersection management," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 374–380, IEEE, 2018.

[94] F. Creemers, A. I. M. Medina, E. Lefeber, and N. van de Wouw, "Design of a supervisory controller for cooperative intersection control using model predictive control," *IFAC-PapersOnLine*, vol. 51, no. 33, pp. 74–79, 2018.

[95] Q. Lu and K.-D. Kim, "A mixed integer programming approach for autonomous and connected intersection crossing traffic control," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pp. 1–6, IEEE, 2018.

[96] E. Steinmetz, R. Hult, Z. Zou, R. Emardson, F. Brännström, P. Falcone, and H. Wymeersch, "Collision-aware communication for intersection management of automated vehicles," *IEEE access*, vol. 6, pp. 77359–77371, 2018.

[97] H. Wei, L. Mashayekhy, and J. Papineau, "Intersection management for connected autonomous vehicles: A game theoretic framework," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 583–588, IEEE, 2018.

[98] F. Saust, J. M. Wille, and M. Maurer, "Energy-optimized driving with an autonomous vehicle in urban environments," in *2012 IEEE 75th Vehicular Technology Conference (VTC Spring)*, pp. 1–5, IEEE, 2012.

[99] L. Makarem and D. Gillet, "Fluent coordination of autonomous vehicles at intersections," in *2012 IEEE international conference on systems, man, and cybernetics (SMC)*, pp. 2557–2562, IEEE, 2012.

[100] S. Adams and M. Rutherford, "Towards decentralized waypoint negotiation," in *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[101] M. N. Mladenović and M. M. Abbas, "Self-organizing control framework for driverless vehicles," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pp. 2076–2081, IEEE, 2013.

[102] G. Lu, L. Li, Y. Wang, R. Zhang, Z. Bao, and H. Chen, "A rule based control algorithm of connected vehicles in uncontrolled intersection," in *17th international IEEE conference on intelligent transportation systems (itsc)*, pp. 115–120, IEEE, 2014.

[103] M. Tlig, O. Buffet, and O. Simonin, "Stop-free strategies for traffic networks: Decentralized on-line optimization.," in *ECAI*, pp. 1191–1196, 2014.

[104] M. Tlig, O. Buffet, and O. Simonin, "Decentralized traffic management: A synchronization-based intersection control," in *2014 International Conference on Advanced Logistics and Transport (ICALT)*, pp. 109–114, IEEE, 2014.

[105] V. Savic, E. M. Schiller, and M. Papatriantafilou, "Distributed algorithm for collision avoidance at road intersections in the presence of communication failures," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1005–1012, IEEE, 2017.

[106] Z. Wang, G. Wu, P. Hao, and M. J. Barth, "Cluster-wise cooperative eco-approach and departure application along signalized arterials," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 145–150, IEEE, 2017.

[107] C. Liu, C.-W. Lin, S. Shiraishi, and M. Tomizuka, "Distributed conflict resolution for connected autonomous vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 1, pp. 18–29, 2017.

[108] J. Wu, A. Abbas-Turki, and A. El Moudni, "Discrete methods for urban intersection traffic controlling," in *VTC Spring 2009-IEEE 69th Vehicular Technology Conference*, pp. 1–5, IEEE, 2009.

[109] F. Yan, M. Dridi, and A. El Moudni, "An autonomous vehicle sequencing problem at intersections: A genetic algorithm approach," *International Journal of Applied Mathematics and Computer Science*, vol. 23, no. 1, pp. 183–200, 2013.

[110] K. Zhang, A. Yang, H. Su, A. de La Fortelle, and X. Wu, "Unified modeling and design of reservation-based cooperation mechanisms for intelligent vehicles," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1192–1199, IEEE, 2016.

[111] Z. Cao, H. Guo, and J. Zhang, "A multiagent-based approach for vehicle routing by considering both arriving on time and total travel time," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 9, no. 3, pp. 1–21, 2017.

[112] F. Perronnet, A. Abbas-Turki, and A. El Moudni, "Vehicle routing through deadlock-free policy for cooperative traffic control in a network of intersections: Reservation and congestion," in *17th international IEEE conference on intelligent transportation systems (ITSC)*, pp. 2233–2238, IEEE, 2014.

[113] F. Perronnet, A. Abbas-Turki, J. Buisson, A. El Moudni, R. Zeo, and M. Ahmane, "Cooperative intersection management: Real implementation and feasibility study of a sequence based protocol for urban applications," in *2012 15th international IEEE conference on intelligent transportation systems*, pp. 42–47, IEEE, 2012.

[114] J. Gregoire and E. Frazzoli, "Hybrid centralized/distributed autonomous intersection control: Using a job scheduler as a planner and inheriting its efficiency guarantees," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 2549–2554, IEEE, 2016.

[115] C. Wuthishuwong and A. Traechtler, "Consensus-based local information coordination for the networked control of the autonomous intersection management," *Complex & Intelligent Systems*, vol. 3, no. 1, pp. 17–32, 2017.

[116] SAE, "J3016b: Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles - sae international." https://www.sae.org/standards/content/j3016_201806/. (Accessed on 07/07/2020).

[117] D. A. Spielman and S.-H. Teng, "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time," *Journal of the ACM (JACM)*, vol. 51, no. 3, pp. 385–463, 2004.

[118] H. G. Santos and T. A. Toffolo, "Mixed integer linear programming with python." https://buildmedia.readthedocs.org/media/pdf/python-mip/latest/python-mip.pdf. (Accessed on 07/07/2020).

[119] Y. J. Li, "An overview of the dsrc/wave technology," in *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, pp. 544–558, Springer, 2010.

[120] J. Erdmann and D. Krajzewicz, "SumoâĂŹs road intersection model," in *Simulation of Urban MObility User Conference*, pp. 3–17, Springer, 2013.

[121] E. Seligman, T. Schubert, and M. A. K. Kumar, *Formal verification: an essential toolkit for modern VLSI design*. Morgan Kaufmann, 2015.

[122] B. Beckert and R. Hähnle, "Reasoning and verification: State of the art and current trends," *IEEE Intelligent Systems*, vol. 29, no. 1, pp. 20–29, 2014.

[123] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, and M. Fisher, "Formal specification and verification of autonomous robotic systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–41, 2019.

[124] S. Bensalem, L. De Silva, A. Griesmayer, F. Ingrand, A. Legay, and R. Yan, "A formal approach for incremental construction with an application to autonomous robotic systems," in *International Conference on Software Composition*, pp. 116–132, Springer, 2011.

[125] A. Katriniok, P. Kleibaum, C. Ress, and L. Eckstein, "Automation of road vehicles using v2x: An application to intersection automation," tech. rep., SAE Technical Paper, 2017.

[126] L. E. Fernandes, V. Custodio, G. V. Alves, and M. Fisher, "A rational agent controlling an autonomous vehicle: Implementation and formal verification," *arXiv preprint arXiv:1709.02557*, 2017.

[127] M. Webster, M. Salem, C. Dixon, M. Fisher, and K. Dautenhahn, "Formal verification of an autonomous personal robotic assistant," in *Proc. Formal Verification Modeling Human-Mach. Syst.: Papers AAAI Spring Symp*, pp. 74–79, AIAA, 2014.

[128] Y. Selvaraj, W. Ahrendt, and M. Fabian, "Verification of decision making software in an autonomous vehicle: An industrial case study," in *International Workshop on Formal Methods for Industrial Critical Systems*, pp. 143–159, Springer, 2019.

[129] S. M. Loos and A. Platzer, "Safe intersections: At the crossing of hybrid systems and verification," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 1181–1186, IEEE, 2011.

[130] M. Hilscher, S. Linker, E.-R. Olderog, and A. P. Ravn, "An abstract model for proving safety of multi-lane traffic manoeuvres," in *International Conference on Formal Engineering Methods*, pp. 404–419, Springer, 2011.

[131] M. Schwammberger, "An abstract model for proving safety of autonomous urban traffic," *Theoretical Computer Science*, vol. 744, pp. 143–169, 2018.

[132] M. Hilscher, S. Linker, and E.-R. Olderog, "Proving safety of traffic manoeuvres on country roads," in *Theories of Programming and Formal Methods*, pp. 196–212, Springer, 2013.

[133] B. Xu and Q. Li, "A spatial logic for modeling and verification of collision-free control of vehicles," in *2016 21st International Conference on Engineering of Complex Computer Systems (ICECCS)*, pp. 33–42, IEEE, 2016.

[134] A. Legay and M. Viswanathan, "Statistical model checking: challenges and perspectives," 2015.

[135] E.-Y. Kang, D. Mu, L. Huang, and Q. Lan, "Verification and validation of a cyber-physical system in the automotive domain," in *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 326–333, IEEE, 2017.

[136] J. Quilbeuf, M. Barbier, L. Rummelhard, C. Laugier, A. Legay, B. Baudouin, T. Genevois, J. Ibañez-Guzmán, and O. Simonin, "Statistical Model Checking Applied on Perception and Decision-making Systems for Autonomous Driving," in *PPNIV 2018 - 10th Workshop on Planning, Perception and Navigation for Intelligent Vehicles*, (Madrid, Spain), pp. 1–8, Oct. 2018.

[137] B. Barbot, B. Bérard, Y. Duplouy, and S. Haddad, "Statistical Model-Checking for Autonomous Vehicle Safety Validation," in *Conference SIA Simulation Numérique*, (Montigny-le-Bretonneux, France), Société des Ingénieurs de l'Automobile, Mar. 2017.

[138] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?," *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.

[139] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen, "Uppaal smc tutorial," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 4, pp. 397–415, 2015.

[140] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "Algorithmic analysis of nonlinear hybrid systems," *IEEE transactions on automatic control*, vol. 43, no. 4, pp. 540–554, 1998.

[141] K. Sen, M. Viswanathan, and G. Agha, "Statistical model checking of black-box probabilistic systems," in *International Conference on Computer Aided Verification*, pp. 202–215, Springer, 2004.

[142] W. J. Thong and M. Ameedeen, "A survey of petri net tools," in *Advanced Computer and Communication Engineering Technology*, pp. 537–551, Springer, 2015.

[143] R. C. Armstrong, R. J. Punnoose, M. H. Wong, and J. R. Mayo, "Survey of existing tools for formal verification," *SANDIA REPORT SAND2014-20533*, 2014.

[144] *Uppaal home page.* `http://www.uppaal.org/`.

[145] R. Alur, T. Feder, and T. A. Henzinger, "The benefits of relaxing punctuality," *Journal of the ACM (JACM)*, vol. 43, no. 1, pp. 116–146, 1996.