Distributed Collaboration of Connected Vehicles at Unsignalized Intersections using Parallel Monte Carlo Tree Search

Seán Hassett

Supervisor: Mélanie Bouroche
August 9, 2021

A dissertation submitted in partial fulfilment of the requirements for the degree of MCS (Computer Science)
Declaration

I hereby declare that this dissertation is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write.

Signed: ___________________________  Date: ___________________________
Abstract

Connected Autonomous Vehicles (CAVs) at unsignalized intersections present an opportunity for greater vehicle throughput and fewer stoppages. By coordinating their arrival times at the intersections, vehicles can avoid collisions while minimising the need to stop completely. Several solutions to this problem have been put forward with some far outperforming signalized intersection models such as the traditional traffic light junction.

These solutions offer a mix of centralised and distributed approaches, with centralised solutions requiring a controller to be physically present at the intersection and distributed solutions involving only the vehicles with no additional infrastructure required.

This paper presents a distributed solution to CAV traversal of unsignalized intersections using a parallel implementation of Monte Carlo Tree Search (MCTS). The root parallel method is used along with a majority vote system in order to unify the search trees. MCTS has shown promise in a centralized intersection management solution, in particular with regards to computation time and handling higher traffic density. With some existing distributed solutions shown to struggle in higher traffic density, we aim to show that MCTS can work in a distributed solution and mitigate the problem of solving for higher traffic density.

We compare the MCTS approach with a basic first-come first-served (FCFS) policy as has been used in existing solutions. The paper will demonstrate that this parallel implementation of MCTS is successful at safely coordinating the vehicles at the intersection and is capable of outperforming the FCFS policy. It is also noted that the root parallel method may not be the most efficient use of the combined computational power of the vehicles as the consensus amongst vehicles on the chosen solution is typically very low.
Acknowledgements

I would like to thank my supervisor Mélanie for her assistance with this project, for allowing me the freedom to direct the project and the support to get it completed.
Contents

Abstract

1 Introduction
  1.1 Motivation ........................................... 1
  1.2 Contribution ........................................... 2
  1.3 Roadmap ............................................... 2

2 Background
  2.1 Connected and Autonomous Vehicles ......................... 4
    2.1.1 Cooperative Driving .............................. 5
    2.1.2 Communication Capability ......................... 6
  2.2 Unsignalized Intersections ................................ 6
    2.2.1 Improving Signalized Intersections .................. 7
  2.3 Intersection Modeling .................................. 8
    2.3.1 Space and Time Discretization ...................... 8
    2.3.2 Trajectory Modeling ................................ 8
    2.3.3 Collision Region Modeling ......................... 8
    2.3.4 Zoning ........................................... 9
  2.4 Centralized VS Distributed Solutions ....................... 11
    2.4.1 Centralized ....................................... 11
    2.4.2 Distributed ..................................... 12
  2.5 Monte Carlo Tree Search ................................ 13
    2.5.1 History and Applications ......................... 13
    2.5.2 Methodology ..................................... 14
    2.5.3 Parallelization .................................. 16
  2.6 Summary ............................................... 18

3 State of the Art ........................................ 19
  3.1 Existing Solutions .................................... 19
    3.1.1 Discrete & Regroup ................................ 19
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.2</td>
<td>Level-k Game Theory</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Virtual Platoon</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Task Area Partition Framework</td>
</tr>
<tr>
<td>3.1.5</td>
<td>Monte Carlo Tree Search</td>
</tr>
<tr>
<td>3.2</td>
<td>Research Objective</td>
</tr>
<tr>
<td>3.3</td>
<td>Summary</td>
</tr>
<tr>
<td>4</td>
<td>Design</td>
</tr>
<tr>
<td>4.1</td>
<td>Overview</td>
</tr>
<tr>
<td>4.2</td>
<td>Zones</td>
</tr>
<tr>
<td>4.3</td>
<td>Passing Order</td>
</tr>
<tr>
<td>4.4</td>
<td>Vehicle Adjustments</td>
</tr>
<tr>
<td>4.5</td>
<td>Monte Carlo Tree Search</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Cost Function</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Special Consideration</td>
</tr>
<tr>
<td>4.6</td>
<td>Decentralizing</td>
</tr>
<tr>
<td>4.7</td>
<td>Heuristic</td>
</tr>
<tr>
<td>4.8</td>
<td>Factors Affecting Performance</td>
</tr>
<tr>
<td>4.9</td>
<td>Security and Privacy Considerations</td>
</tr>
<tr>
<td>4.10</td>
<td>Summary</td>
</tr>
<tr>
<td>5</td>
<td>Implementation</td>
</tr>
<tr>
<td>5.1</td>
<td>Software Overview</td>
</tr>
<tr>
<td>5.1.1</td>
<td>SUMO</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Netedit</td>
</tr>
<tr>
<td>5.1.3</td>
<td>TraCI</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Python TraCI</td>
</tr>
<tr>
<td>5.2</td>
<td>Python Implementation</td>
</tr>
<tr>
<td>5.2.1</td>
<td>SUMO Network</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Vehicle Class</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Passing Order Class</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Monte Carlo Tree Search</td>
</tr>
<tr>
<td>5.2.5</td>
<td>Reporting</td>
</tr>
<tr>
<td>5.3</td>
<td>Summary</td>
</tr>
<tr>
<td>6</td>
<td>Evaluation</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion</td>
</tr>
<tr>
<td>7.1</td>
<td>Future Work</td>
</tr>
</tbody>
</table>
List of Figures

2.1 From Chen and Englund [2016]. Illustration of intersection discretization. Intersections are discretized into a grid of tiles (cells). (a) A four-way intersection with single straight traffic from each direction. The intersection is discretized into 20 tiles. (b) A four-way intersection with straight, left-turn and right-turn traffic. The discretization is in more detail with 64 tiles. (c) A roundabout discretized into 24 tiles. .............................................. 9

2.2 From Chen and Englund [2016]. Trajectory illustration. (a) A four-way intersection with no turns allowed, each entrance is associated with one trajectory, i.e. a straight travel route. (b) A four-way multi-lane intersection, each entrance is associated with three trajectories, i.e. straight, left-turn and right-turn. (c) A typical roundabout, where each entrance is associated with four trajectories, i.e. right-turn, straight, left-turn and U-turn. .............. 10

2.3 From Chen and Englund [2016]. Illustration of collision regions. Intersections are modeled by combining the space discretization and trajectory modeling, where only the collision regions need to be considered. Instead of discretizing the whole intersection, there are in total 16 collision regions that need to be reserved and allocated. .............................................. 11

2.4 From Bian et al. [2020]. Illustration of a four-way intersection showing the different zones present. Inbound traffic passes through the Observation Zone, Optimization Zone and Control Zone in order before traversing the Merging Zone, thereby traversing the intersection. ......................... 12

2.5 Intersection zoning and space discretization of Xu et al. [2020]. ........... 13

2.6 Intersection zoning of Qian et al. [2020]. .................................. 14

2.7 From Browne et al. [2012]. A general MCTS iteration. ...................... 15

2.8 From Chaslot et al. [2008]. Leaf Parallelization of MCTS. ................... 16

2.9 From Chaslot et al. [2008]. Root Parallelization of MCTS. ................... 17

2.10 From Chaslot et al. [2008]. Tree Parallelization of MCTS. ................. 18

3.1 From Qian et al. [2020], illustration of group arrival at intersection. ...... 19

3.2 Projection of inbound vehicles onto a virtual lane, from Xu et al. [2018]. .. 21
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3</td>
<td>Results from Xu et al. [2020] showing performance vs FCFS.</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Intersection zones. Control Zone shown in green with Intersection Zone shown in grey.</td>
<td>25</td>
</tr>
<tr>
<td>4.2</td>
<td>MCTS selection step. Yellow vehicles indicate eligibility for selection.</td>
<td>29</td>
</tr>
<tr>
<td>5.1</td>
<td>SUMO Interface.</td>
<td>33</td>
</tr>
<tr>
<td>5.2</td>
<td>Netedit Interface.</td>
<td>34</td>
</tr>
<tr>
<td>5.3</td>
<td>Vehicles in various states during the simulation.</td>
<td>38</td>
</tr>
<tr>
<td>6.1</td>
<td>Graph comparing total time of MCTS, FCFS and TLS.</td>
<td>42</td>
</tr>
<tr>
<td>6.2</td>
<td>Graph comparing total adjustment required of MCTS and FCFS.</td>
<td>43</td>
</tr>
<tr>
<td>6.3</td>
<td>Graph comparing standard deviation in Adjust of MCTS and FCFS.</td>
<td>44</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Solutions for Unsignalized Intersections ........................................... 22
6.1 Evaluation results ................................................................................. 42
1 Introduction

In this chapter, Section 1.1 will discuss the motivations of the paper and set out the goals we hope to achieve. Section 1.2 will outline the contribution that this paper makes to the existing literature while Section 1.3 will provide an overview of the layout of the paper.

1.1 Motivation

Intersections pose a problem for traffic flow as they involve overlapping routes which opens up the possibility of traffic collisions. Signalized intersections avoid collisions by signaling vehicles when it is safe for them to enter the intersection. Typically, signalized intersections are not very sophisticated and run on a fixed schedule, leading to inefficient traffic flow which leads to traffic delays and poor fuel economy which has effects on emissions as well as a financial impact on the vehicle owner.

Unsignalized intersections by contrast don’t explicitly signal when a vehicle can enter the intersection but rather have a set of rules by which drivers can discern when they can enter. Unsignalized intersections carry a great risk of collision but can offer improvements to traffic throughput and minimise driver delay.

Connected Autonomous Vehicle (CAV) traffic has the potential to further increase traffic throughput utilizing the superior capabilities of CAVs as compared to human drivers. CAVs are capable of communicating with each other and cooperating at intersections so as to pass in a more efficient manner. Distributed solutions for intersection control have the potential to achieve these benefits with no additional infrastructure required at the intersection which removes a potential cost factor.

The motivation for this paper was to assess the state of art solutions to intersection management with a particular focus on distributed solutions which are called essential by Eskandarian et al. [2021] in order to deal with imperfections in Vehicle-to-Infrastructure (V2I) and Vehicle-to-Vehicle (V2V) communication. Our goal was to identify an area in which we could expand on the existing literature by exploring a new solution which could improve upon existing solutions in some way.
1.2 Contribution

This paper provides an overview of the state of the art in Connected Autonomous Vehicle technology and the applications which they can serve. We also look more specifically at applications in cooperative intersection management at unsignalized intersections, providing a sample of solutions from recent years and a summary of the specific problems these solutions look to solve.

This paper also provides a design for a distributed cooperative intersection management solution using Monte Carlo Tree Search which has proven effective in a centralized approach. This solution is shown to be more effective than a first-come first-served policy that some existing solutions opt for and shows promise as the optimisation method within a larger intersection management system.

Finally, the paper proposes a path for future work in improving the intersection management solution, with a particular focus on how to more efficiently make use of the combined processing power available to the vehicles at the intersection.

1.3 Roadmap

Chapter 2 of this paper will give background information on the technologies and concepts that will be examined and utilised throughout the paper. We will look at some existing applications of these technologies and the benefits they can bring over traditional systems. Following this, Chapter 3 will examine the current state of the art in the area of intersection management systems. A sample of existing solutions will be looked at in detail, focusing on the particular challenge each solution aims to achieve.

Chapter 4 will then give an overview of our design for an intersection management system. We will outline the specific niche our solution is designed to fill as well as examining how our design relates to existing solutions and what it does differently. We will also outline the assumptions that will be made and look at what scenarios we expect it to work well in. Chapter 5 will then describe our implementation of this solution. We will look at the software we used to simulate the solution and the pros and cons to that approach. We will also outline how we implemented specific parts of our design within this framework and where there might be limitations. Finally we will describe how extract metrics from the implementation in order to evaluate its performance.

Chapter 6 will look at results generated from the implementation, explaining which metrics are used and why and summarising the results from multiple scenarios in tabular form. Results from alternative intersection management systems are also shown for the sake of comparison. Chapter 7 will draw conclusions from these results and analyse where our
design and implementation performs well and where it could be improved. An overall conclusion of the suitability of our solution to the domain of intersection management and the promise shown by the solution will be given. This chapter also provides suggestions for future work to build upon the work of this paper.
2 Background

This chapter provides background information on the concepts that we will be dealing with. Section 2.1 gives an overview of Connected and Autonomous Vehicles (CAVs). We look at what defines a CAV, what are some of the typical applications they can be used for and what benefits they offer compared to the human driver alternative. Section 2.2 then looks at Unsignalized Intersections, how they differ from their Signalized counterparts, what are the primary challenges that are to be solved at unsignalized intersections including the challenge we are interested in, Autonomous Intersection Management systems. Finally we examine the benefits can be drawn from solving these challenges.

In Section 2.3 we look at ways an intersection can be modeled in order to facilitate different approaches to intersection traffic management. Three examples of intersection modeling are presented followed by a sample of approaches to dividing the area around the intersection into zones. Section 2.4 will provide an overview of the differences between centralized and decentralized approaches to intersection management, including the motivations for each approach as well the pros and cons of each.

Section 2.5 will then provide an overview of Monte Carlo Tree Search (MCTS). This is an algorithm that has been used in intersection management and shows promise as a method of organising traffic flow. This section will provide a short history of MCTS and the applications it has been used for as well as giving an overview of the general methodology of MCTS and how it works.

2.1 Connected and Autonomous Vehicles

Autonomous Vehicles (AVs) are vehicles capable of self-driving without human driver input. A Connected Autonomous Vehicle (CAV) is an AV which has some degree of connectivity. A common example of connectivity is the ability of vehicles to use GPS for navigation. IEEE [2012] estimate that by 2040, 75% of all traffic will be comprised of CAVs.

Eskandarian et al. [2021] performed a study examining the current state of CAV technology in terms of the sensors available to them and some of the perception, planning and control
algorithms they can utilize. The study claims that 94% of traffic accidents are caused by human error and factors such as drowsiness, distraction and decision making. Thus CAVs can effect a large reduction in the number of accidents by virtue of being immune to these types of errors. Through their connectivity, CAVs can also achieve greater route optimization which leads to more efficient fuel economy and a greater level of traffic throughput. CAVs are also more capable of making fine adjustments to their trajectory compared to human drivers which allows for more precise solutions to be implemented.

Section 2.1.1 will look at different applications of Cooperative Driving that are made possible by CAV technology. Section 2.1.2 will then look at how the communication capabilities of CAVs can evolve over time enabling different applications and take a short look at what today’s technology enables.

2.1.1 Cooperative Driving

Eskandarian et al. [2021] point out applications in cooperative driving that are made possible by CAV ability to communicate with other CAVs. The primary fields they point out are Cooperative Adaptive Cruise Control, Cooperative Perception and Cooperative Intersection Control. These applications and their uses are discussed in further detail below.

Cooperative Adaptive Cruise Control (CACC)

CACC is used to allow for a platoon of vehicles to travel in the same trajectory in a more efficient manner than human drivers would be able to manage. CAVs can communicate with vehicles that are following them to forewarn of changes in velocity so that these changes can be executed in parallel rather than the ripple effect we see with human drivers. This allows platoons of CAVs to drive with shorter gaps between them than would be safe for humans as there is no reaction time for the CAVs.

Cooperative Perception

Cooperative perception involves CAVs sharing their sensor data with neighbouring CAVs to effectively increase the line of sight of the group of CAVs. This again exceeds human capability as a human driver at the front of a platoon of vehicles may have no idea of what is happening at the back of the platoon. With cooperative perception, the vehicle at the front can receive the same perception data as the vehicle at the back and vice versa, increasing the awareness of all the CAVs in the platoon.

An example of where this would be beneficial would be the avoidance of an obstacle on the road. While the leading vehicle can see the obstacle coming, the following vehicles don’t see it until the leading vehicle is already moving around it. With cooperative perception, all of
the following vehicles can be made aware of the upcoming obstacle as soon as the leading vehicle becomes aware of it.

**Cooperative Intersection Control**

CAVs also provide an opportunity for more efficient traversal of intersections. Intersections pose a problem as there will be overlapping routes through the intersection and thus the possibility of collisions. CAVs can communicate with each other on approach to an intersection in order to traverse the intersection while avoiding collisions, maximizing throughput and minimizing vehicle delay. There are many centralized and decentralized intersection control methods already developed although Eskandarian et al. [2021] point out that the majority of these are only tested in simulations rather than in real traffic. In order to function in real traffic, a method would have to also be able to factor in human traffic as well.

### 2.1.2 Communication Capability

While Eskandarian et al. [2021] point out that many cooperative intersection solutions assume perfect communication between vehicles and state that this assumption would not carry forward to a real world application, with it being some time off before CAV traffic becomes dominant on the roads it’s arguably difficult to say what capabilities CAVs will have when they will be implementing these solutions in the real world. Coronado et al. [2019] provide an overview of 5G connectivity and the effects it will have on CAVs. Specifically, they demonstrate how certain tasks of the CAV such as lane tracking can be offloaded to the cloud using a Multi-access Edge Computing enabled network and how network slicing allows for the creation of virtual networks that are specialized for a specific requirement such as low latency.

### 2.2 Unsignalized Intersections

An unsignalized intersection is an intersection in which vehicles are not given explicit instructions as to when they should enter the intersection, rather they follow rules such as right-of-way in order to safely navigate the intersection. An example of a signalized intersection would be a traffic light junction, in which drivers are signalled when they can enter the intersection by the colour of their respective traffic light. An example of an unsignalized intersections would be those governed by right-of-way rules, such as an intersection between a major and a minor road where vehicles coming from the major road have priority over those coming from the minor road. These intersections are usually managed by Stop and Yield signs which indicate the rules to the approaching vehicles. Another example of an unsignalized intersection would be a roundabout, where vehicles
follow a rule of thumb in order to traverse the intersection i.e. give priority to traffic from a certain direction.

Belkhouche [2019] points out an area of interest in intersection management is the Autonomous Intersection Management (AIM) system. AIM systems are designed to increase vehicle throughput at an intersection by reducing unnecessary stoppages that may be seen in a traffic light system and optimising traffic flow. Reducing stoppages has a positive impact on fuel efficiency of vehicles as it is more efficient for vehicles to slow down and accelerate again than it is for them to stop completely and have to accelerate from zero again. Furthermore, vehicles can waste fuel while idling at an intersection. AIM systems generally work by having vehicles communicate with a controller in order to reserve a time at the intersection. The system will then organise arrival times in such a way that collisions are avoided.

Unsignalized intersections generally perform better than traditional signalized intersections in terms of fuel efficiency and traffic delay. The disadvantage is that they are potentially more prone to collisions and accidents if it isn’t fully clear which vehicle has the priority. AIM systems can aid in this as well by giving explicit instructions to vehicles as to when they can enter the intersection and thus avoiding potential collisions.

2.2.1 Improving Signalized Intersections

As well as creating AIM systems to optimise traffic flow at intersections, efforts have been made to improve upon signalized intersection systems. Younis and Moayeri [2017] propose a dynamic traffic light system which uses a network of sensors to control traffic light signals as opposed to the typical method of having the lights on a fixed schedule. They supply two design approaches, the first of which uses sensors in the road networks to gather data and the second of which places the sensors on the vehicles. Both designs offer a traffic light controller which can smartly schedule the traffic light based on the traffic demands and thus reduce vehicle waiting time while increasing vehicle throughput.

Zhang et al. [2018] also propose a Virtual Traffic Light system that is designed to replace physical traffic lights which removes the cost factor associated with traffic light systems. The vehicles collaborate on the scheduling of the traffic light system, with the signals sent directly to the vehicles rather than shown on physical lights at the intersection. This solution includes more efficient scheduling than traditional traffic light systems that run on fixed schedules.
2.3 Intersection Modeling

There are several ways of modeling an intersection in order to design a solution for collision avoidance. Chen and Englund [2016] examine three approaches while we also see the concept of zoning in the work of Bian et al. [2020], Xu et al. [2020] and Qian et al. [2020]. These three approaches are Space and Time Discretization, Trajectory Modeling and Collision Region Modeling and are examined in Section 2.3.1, Section 2.3.2 and Section 2.3.3.

Section 2.3.4 will look at different ways of using zones at the intersection, where the areas on approach to the intersection are designated into different zones with different purposes for the solution. We will look at how some existing intersection management systems apply the concept of zones at the intersection.

2.3.1 Space and Time Discretization

With this approach, the intersection is divided into spatial regions, typically a grid. The collision avoidance problem is then transformed into a resource allocation and optimization problem, with time slots and the discretized regions of the intersection being allocated to the vehicles traversing the intersection.

The granularity of the discretized regions has an effect on the quality of the solution as well as computational complexity. In general, higher granularity i.e. dividing the intersection into smaller regions, can result in a better solution but also causes higher complexity. Thus, finding an optimal level of granularity becomes part of the solution.

2.3.2 Trajectory Modeling

This approach involves defining the intersection as a set of trajectories that vehicles can follow as they traverse the intersection. Solutions based on trajectory modeling then involve finding combinations of non-conflicting trajectories, which we call safe patterns.

Vehicles which adhere to a safe pattern are allowed to traverse the intersection as normal while vehicles whose trajectories will overlap with another must alter their trajectory. One way of solving conflicting trajectories is to factor in vehicle control parameters. For example, a vehicle can be instructed to decelerate so that their trajectory no longer overlaps with any others and they can traverse the intersection in a safe pattern.

2.3.3 Collision Region Modeling

Since the only areas of interest to us in an intersection are the points where trajectories can overlap, another way of modeling the intersection is consideration of only the potential
2.3.4 Zoning

In addition to the modeling techniques outlined above, some solutions categorise different zones at the intersection. In the approach used by Bian et al. [2020] we see four different zones which the vehicles pass through as they approach and then traverse the intersection.

The first of these is the Observation Zone, where the vehicles observe the states of all other vehicles at the intersection. Following this is the Optimization Zone, where the computation
Figure 2.2: From Chen and Englund [2016]. Trajectory illustration. (a) A four-way intersection with no turns allowed, each entrance is associated with one trajectory, i.e. a straight travel route. (b) A four-way multi-lane intersection, each entrance is associated with three trajectories, i.e. straight, left-turn and right-turn. (c) A typical roundabout, where each entrance is associated with four trajectories, i.e. right-turn, straight, left-turn and U-turn.

of an optimal trajectory is performed. Next we have the Control Zone, where the vehicles alter their trajectory to the computed solution using the vehicle's controls. Finally we have the Merging Zone, which is the centre of the intersection where the collision regions exist. It is a common assumption that vehicles pass through this zone uniformly having already adjusted their trajectories to avoid collisions, as seen in the work of Bian et al. [2020] and Xu et al. [2020]. By contrast, the solution from Qian et al. [2020] has a group of vehicles time their arrivals to the intersection so that they all arrive at the same time and then optimise their paths around each other as they traverse the intersection.

In Figure 2.5 we see the approach of Xu et al. [2020] into using zones at the intersection. Their approach is simpler than that of Bian et al. [2020] with a single control zone in which the solution is implemented. Traffic at the intersection is uniform so the control zone boundary ends at the entrance to the intersection. We can also see them employ space discretization of the intersection.

Figure 2.6 shows the zoning approach used by Qian et al. [2020]. They employ a Grouping Zone in order to group vehicles based on their distance from the intersection. Vehicles move
2.4 Centralized VS Distributed Solutions

Any collision avoidance solution for unsignalized intersections can be classified as being centralized or decentralized. Chen and Englund [2016] provide a summary of the principal differences between centralized and distributed solutions and the challenges faced by both.

2.4.1 Centralized

A centralized solution uses a single controller in order to compute optimal trajectories for passing vehicles at the intersection. Each vehicle communicates with the controller to convey its current state and all computation is performed by the controller which relays the calculated trajectories to each vehicle. One of the advantages of centralized solutions is that they can handle more complex computation because you only need one powerful controller to perform the computation, as opposed to needing many processors capable of heavy computation. The work done by the vehicles is reduced to communication only. The central controller is also capable of utilising historical data to complement its computation.

The main drawback of a centralized system is the need to have physical infrastructure installed at the intersection. This introduces an initial capital cost as well as the cost of maintaining the controllers. Centralized systems also suffer from having a single point of
failure. If the controller fails, the vehicles will need to have some kind of fallback mechanism in order to safely navigate the intersection.

2.4.2 Distributed

In decentralized or distributed solutions, there is no central controller. Instead, all of the computation is performed by the vehicles as they traverse the intersection. The primary benefit of distributed solutions is the removal of the central controller. This leads to reduced costs with no need to install and maintain infrastructure at each junction. It also eliminates a potential issue with the controller being a single point of failure. The downside to distributed solutions is that it shifts a lot of work to the vehicles. Vehicles need to be capable of complex computation with solutions needing to be calculated in a short amount of time. As well, the amount of communication required is increased, with every vehicle needing to communicate with every other vehicle leading to potentially high bandwidth requirements.
2.5 Monte Carlo Tree Search

In this section we will look at Monte Carlo Tree Search (MCTS). In Section 2.5.1 we will look at the history of MCTS and how it came to prominence. We will also look at some of the classic applications of MCTS and why it works particularly well for those applications. Section 2.5.2 will provide an overview on the methodology of MCTS. We will look at how the algorithm works in a general sense as well as analysing each of steps involved in one iteration of a MCTS solution. Finally Section 2.5.3 will examine some of the ways MCTS can be parallelized and how these approaches might be useful in employing MCTS in a distributed intersection management system.

2.5.1 History and Applications

Monte Carlo Tree Search (MCTS) is a best-first search method which uses random sampling to build a search tree. MCTS first gained popularity in the games industry, being used in solutions for games such as Scrabble and Bridge. Browne et al. [2012]. MCTS became more prominent in the game of Go, with many of the early Go computers capable of beating humans utilising MCTS.

Go is an incredibly complex game with a huge number of possible game states. The complexity of a game can be approximated by taking the number of possible moves per position, \( b \) and the length of the game, \( d \) and solving \( b^d \) which gives you the total number
of possible move sequences. In chess, a game where exhaustive search solutions aren’t feasible, these values are $b \approx 35$ and $d \approx 80$. In Go on a full-sized board they are $b \approx 250$ and $d \approx 150$. Silver et al. [2016].

By 2012, some Go computers utilising MCTS solutions were able to beat human opponents on reduced-size boards. In 2015, the computer program AlphaGo defeated a professional Go player, Fan Hui, on a full-sized board, becoming the first computer program to win against a human on a full-sized board. The following year, AlphaGo defeated a top-ranked professional Go player, Lee Sedol, 4 games to 1 in a prominent 5 game series. AlphaGo uses a combination of deep neural networks with MCTS. Silver et al. [2016].

MCTS has also been applied to domains outside of games. Browne et al. [2012] outline applications in combinatorial optimization problems, constraint satisfaction problems, scheduling problems, sample-based planning problems and procedural content generation. In 2020, Xu et al. [2020] presented a solution for unsignalized intersection traversal using MCTS. In 2015, Golpayegani et al. [2015] presented a decentralized solution for electricity demand management using a parallel implementation of MCTS.

2.5.2 Methodology

MCTS works by iteratively building a search tree with each node of the tree representing a state of the system in question, e.g. in Go programs, a node would represent a certain
configuration of pieces on the board, with the child nodes from that node being the new states the board would be in given the next move. Nodes have an associated value which is used in assessing the most promising nodes. A node is considered expandable if it represents a non-terminal state. Iterations run until a computational budget is reached, with the budget being given as time or other resources. When the budget is reached, the best root action is returned. Browne et al. [2012]

There are four steps to each iteration.

- **Selection.** Starting at the root node, the tree is traversed via a given policy to select the most urgent expandable node.
- **Expansion.** One or more child nodes are added to expand the tree.
- **Simulation.** A simulation is run from the new node(s) to produce a result.
- **Backpropagation.** The results from the simulation are backpropagated through the selected nodes and their values are updated.

Two policies govern these four steps. The tree policy is used in the selection and expansion steps, to determine what is the most urgently expandable node and to add a child node to the selected node. The default policy is used in the simulation step to play out the non-terminal state to a terminal state and produce a value as a result.

MCTS is capable of running with no domain-specific knowledge. All that is needed are the next legal states for each node and what determines a terminal state. Heuristics can be incorporated into a MCTS in order to make the selection and expansion steps more efficient.

Two policies govern these four steps. The tree policy is used in the selection and expansion steps, to determine what is the most urgently expandable node and to add a child node to the selected node. The default policy is used in the simulation step to play out the non-terminal state to a terminal state and produce a value as a result.

MCTS is capable of running with no domain-specific knowledge. All that is needed are the next legal states for each node and what determines a terminal state. Heuristics can be incorporated into a MCTS in order to make the selection and expansion steps more efficient.

![Tree Policy](image1)

**Figure 2.7:** From Browne et al. [2012]. A general MCTS iteration.

The value of a node is determined by the application. For example in the Go program, the value of a node might be what percentage of simulated games the computer won from that
position. Thus, by selecting the root node with the highest value, the computer chooses the
move that transforms the board to the state where the computer’s wins in the simulated
games were maximised.

Node values can also be used to inform the selection process allowing the program to use its
limited budget to only explore the most promising possibilities. However, there is an
exploration-exploitation dilemma to bear in mind, where the program can search too
narrowly down one path at the expense of other, possibly more promising paths. The
popular method of handling this dilemma is the Upper Confidence Bounds approach
proposed by Kocsis and Szepesvári. Browne et al. [2012].

2.5.3 Parallelization

MCTS can be parallelized by running simulations concurrently via multiple processors or
multiple agents. This can allow more simulations to be run within a given time budget, but
it introduces complexity when combining the independent results into a single search tree.
Browne et al. [2012] describe different ways of parallelizing MCTS.

Leaf Parallelization - This involves running concurrent simulations when a new leaf node
is reached in the search tree. It allows for more better statistics to be gathered by virtue of
running more simulations. Chaslot et al. [2008] point out some problems with leaf
parallelization. One such problem is that the program needs to wait until all threads have
finished before moving on from the simulation step. If the simulations take different
amounts of time, the time for the simulation step will be the longest time of all simulations
run by the threads, thus the average time for the simulation step is likely to be increased.
There is a also an inefficiency caused by threads not sharing information. If some of the
threads have finished their simulation and the results are poor, thus rendering that node
unlikely to be selected for further exploration, the remaining threads will complete their
simulation redundantly, wasting computational resources.

![Leaf parallelization](image)

Figure 2.8: From Chaslot et al. [2008]. Leaf Parallelization of MCTS.

Root Parallelization - This involves building multiple search trees simultaneously. Each
thread or agent performs their own MCTS solution, each working on their own tree and selecting and expanding nodes independent from the other threads. The best root action is determined by combining the results from all of the trees. Soejima et al. [2011] put forward majority voting as a reliable way of obtaining an optimal result from a root parallel MCTS solution. This is contrary to the standard approach of summing the results across all trees, where instead each thread votes for the root action it found to be best and the root action with the most votes is selected.

Golpayegani et al. [2015] proposed an information sharing version of Root Parallel MCTS which they call Collaborative Parallel MCTS (CP-MCTS). In this version, threads periodically share results from simulations with each other in order to improve the quality of each search tree.

![Figure 2.9: From Chaslot et al. [2008]. Root Parallelization of MCTS.](image)

**Tree Parallelization** - Proposed by Chaslot et al. [2008], this involves multiple agents or threads performing MCTS iterations on the same search tree. This allows for greater exploration of the tree in the same amount of time as compared to having a single thread. It introduces problems with managing access to the tree across multiple threads so as to avoid the backpropagated results of one thread to be overwritten by another thread. This can be done using a global mutex on the whole tree, which threads request when they want to backpropagate their results. It is also possible for each node to have its own local mutex instead of a global mutex.

Another issue with tree parallelization is the possibility for multiple threads to traverse very similar paths instead of diversifying their exploration. The proposed solution to this problem is the introduction of a "virtual loss" mechanic. This works by decreasing the value of a node when it is visited by a thread, thus discouraging other threads from also visiting unless the value of the node is high enough that it offsets the virtual loss.
2.6 Summary

This chapter provided an overview of the technologies and concepts that will be utilized in our design. We examined the role of Connected Autonomous Vehicles in traffic management and how they can be utilised to enhance traffic flow and decrease the rate of accidents. We also looked at Unsignalized Intersections and the challenges they pose relative to Signalized Intersections, as well as the benefits they can allow including increased vehicle throughput and a reduction in traffic delays and stoppages.

We looked at different ways an intersection can be modeled in order to design an intersection management solution and how some of these models can be more efficient than another. We also looked at the different ways an intersection can be divided into zones in order to separate out the different elements of a management system and decompose the system into tasks based on distance from the intersection. We then looked at the primary difference between centralized and distributed systems as well as the tradeoffs involved in choosing one or the other.

Finally, we gave an overview of Monte Carlo Tree Search, which we are using in our own intersection management system. We looked at the applications that MCTS has outside of traffic management and the types of problems that it excels at. We also looked at the methodology of MCTS and finally we looked at ways it can be parallelized in order to be incorporated into a distributed solution which is our goal.
3 State of the Art

Section 3.1 of this chapter will give a sample of existing intersection management solutions from recent years, looking at a diverse set of approaches to the problem. Section 3.2 will follow by stating our own research objective for this paper, outlining how our solution relates to the existing solutions mentioned and the gap that we are aiming to fill.

3.1 Existing Solutions

This section will examine a sample of existing intersection management systems. Sections 3.1.1 to 3.1.5 will each examine a system in detail. These solutions are summarised in Table 3.1.

3.1.1 Discrete & Regroup

Presented by Qian et al. [2020], the goal of this solution is to simplify the computation process by grouping vehicles based on their inbound lanes and solving for each group. This means that with a 4-way intersection with one inbound lane per direction, the maximum number of vehicles that will need to be solved for at a time will be 4. While the vehicles within a group may arrive out of order relative to their original projected arrival order to the intersection, the groups themselves will arrive at the intersection in order. This means that this solution is mostly based on a first-come first-served policy, which is marked as FCFS* in Table 3.1. This arrival order is illustrated in Figure 3.1.

Figure 3.1: From Qian et al. [2020], illustration of group arrival at intersection.
3.1.2 Level-k Game Theory

This solution from Nan et al. [2018] is based on the assertion that game theoretical approaches to unsignalized intersection management avoid extreme cases and result in more realistic solutions. A branch of game theory called level-k game theory seeks to model human behaviour and this is what is used to build an autonomous vehicle controller capable of navigating an unsignalized intersection. The solution relies on observation of the behaviour of other vehicles at the intersection and adapting accordingly. The connected element of CAVs is less prevalent here as the vehicles rely on the same physical observations that a human driver would be able to make, as opposed to broadcasting precise data relating to velocity and trajectory etc. The solution is distributed in that it accounts for a single autonomous vehicle, so there is no central controller required. A network of vehicles each implementing this solution with human driver traffic mixed in would be capable of traversing intersections without centralised control.

3.1.3 Virtual Platoon

This solution from Xu et al. [2018] presents the idea of a virtual platoon, with vehicles from all incoming lanes at the intersection being projected onto a single virtual lane, as shown in Figure 3.2. With the virtual platoon formed, a directed graph is created to model the conflicting trajectories between vehicles in the platoon. A depth-first search algorithm is then used to create a depth-first spanning tree, with the theory being that vehicles at the same depth in the spanning tree do not have conflicting trajectories. This means that the vehicles can arrive in the order according to their depth in the tree without risking collision. A distributed controller is designed to facilitate the implementation of the spanning tree solution.

3.1.4 Task Area Partition Framework

This solution by Bian et al. [2020] offers a framework for distributed solutions at unsignalized intersections by discretizing the tasks that need to be performed and solving each task in isolation. They identify three primary tasks:

- Observation - This task involves the process of the vehicles exchanging information with each other as they approach the intersection.

- Optimisation - This task involves optimising the trajectories of vehicles through the intersection in order to facilitate throughput and minimize delays.

- Control - This task involves the vehicles making adjustments to their trajectories in order to meet the optimisation solution computed in the previous task.
3.1.5 Monte Carlo Tree Search

Xu et al. [2020] present a solution using Monte Carlo Tree Search in order to model traffic at the intersection. A search tree is created with the nodes of the tree representing vehicles within the control zone and the hierarchy of the tree indicating the passing order of the vehicles. MCTS is used to find the optimal passing order by expanding the nodes within a given time budget and using a cost function to score pathways through the tree. The cost of a passing order is given as the total delay for all vehicles that would occur in enforcing that passing order. This paper compares the MCTS to a first-in first-out policy - analogous to a first-come first-served (FCFS) policy - in varying levels of traffic density. The MCTS solution performs favourably compared to the FCFS policy, in particular at higher traffic densities, as shown in Figure 3.3.

3.2 Research Objective

We will be looking at optimising traffic flow of Connected Autonomous Vehicles at Urban Unsignalized Intersections. Specifically we will be looking at a distributed solution in which all optimisation and computation is performed by the vehicles. As stated in Section 2.4,
distributed solutions require no additional infrastructure at the intersection. This eliminates a potential cost factor and significantly eases rollout of the system by removing the physical aspect.

Bian et al. [2020] presented a framework for distributed solutions in which the overall solution is decomposed into tasks that can be individually implemented and improved upon. One of these tasks is the optimisation step which is what we are examining. In the paper from Bian et al. [2020], they use a first-come first-served policy in the optimisation task. The results indicate that this policy performs reasonably well but begins to struggle in higher levels of traffic density. In their conclusion they cite the FCFS policy as an area of improvement for future work.

Xu et al. [2020] examined a centralised solution using Monte Carlo Tree Search (MCTS). This solution showed promising results, with computation time in particular being very low, resulting in greater performance in higher traffic densities. With the excellent performance in higher traffic density, MCTS shows promise as a substitute for the FCFS policy used by Bian.
et al. [2020] in their framework. We will investigate whether a parallel implementation of MCTS can be successfully implemented in a distributed solution and whether we can carry over the performance to be able to exceed that of a FCFS policy in a distributed system.

We will make assumptions that are typically made in these solutions. First we will assume perfect communication is possible between vehicles, with no latency or loss. Second, we will assume that all vehicles are uniform in terms of shape and performance, and are capable of perfectly following their programmed routes. Finally, while with unsignalized intersections there are other uncertainties that can impact the performance of a solution and there are Vulnerable Road Users to account for, our solution will not be taking these into consideration.

Performance of the solution will be measured using metrics that are typical to this area. Following from the work of Xu et al. [2020], the most important metric will be the delay in vehicles traversing the intersection, with the goal being to minimise the average travel time of the vehicles. Traffic density is also an important factor as it has a significant impact on computation complexity. Therefore, these metrics can be measured at varying levels of traffic density to measure the impact of increasing density.

In order to evaluate the suitability of MCTS as a method of optimising traffic at unsignalized intersections, its performance will be measured against a FCFS policy as done by Xu et al. [2020] and as used by Bian et al. [2020]. We will also measure performance compared to a traffic light system to measure the efficiency of our solution compared to a classic signalized system.

3.3 Summary

In this Chapter we gave an outline of existing intersection management systems and the ways in which they differ. We went into detail on a sample of recently published systems and elaborated on the main goals of each system. We presented a summary of these solutions in tabular form illustrating their primary method and which ones take a distributed approach.

We then stated our own research objective with regard to these existing solutions, stating our objective as being to improve upon the optimisation method used by Bian et al. [2020] by looking at the centralized solution of Xu et al. [2020] and examining ways in which it could be modified to become part of a distributed solution.
4 Design

This Chapter provides an overview of our own design for an intersection management system. Section 4.1 provides a high-level description of our design and our motivation for this particular approach. Section 4.2 illustrates how we applied zoning at the intersection, as discussed in Section 2.3.4. Section 4.3 describes the passing order which is used in the system to schedule vehicle arrival at the intersection while Section 4.4 describes how the vehicles use the passing order to adjust their trajectory as they approach the intersection.

Section 4.5 describes how we utilise Monte Carlo Tree Search (MCTS) in our design, describing the cost function that we employ in order to give values to nodes as well as illustrating how our particular use case differs from some more traditional applications of MCTS. Section 4.6 describes how we parallelize MCTS in order to implement it in a distributed system without any central controller.

Section 4.7 describes the heuristic that we apply to our MCTS implementation in order to improve its performance. Section 4.8 will look at some of the factors and variables than can have an impact on the overall performance of the system. Finally, Section 4.9 will look at some security and privacy concerns that our system may be vulnerable to and how these might be mitigated.

4.1 Overview

The goal of our design is to take the MCTS solution of Xu et al. [2020] and remove the central controller element. In other words, we want a distributed solution in which the vehicles are capable of working in collaboration to compute a solution using MCTS. MCTS was chosen due to its demonstrated ability to scale into higher traffic densities which was a blocking factor in other solutions.

The solution is time based, with vehicles executing the solution on a regular interval after a given time delta. The duration of this interval is a significant factor in the performance of the solution and is discussed more in Section 4.8. This allows a continuous flow of traffic to
continually calculate efficient paths through the intersection. This also means that a single vehicle may be involved in multiple solutions with different sets of vehicles.

4.2 Zones

We follow the approach used by Xu et al. [2020] in their MCTS solution in order to zone the intersection. As the vehicles approach the intersection, they will enter the Control Zone, wherein they begin to compute the solution. They begin with broadcasting their data to every other vehicle in the Control Zone. This data includes their current position, their intended trajectory through the intersection and their current speed, which can be used in conjunction with their position to calculate their arrival time to the intersection.

The optimal passing order, see Section 4.3, is computed in the Control Zone and the vehicles make adjustments to their trajectory in order to meet this passing order. When the vehicles enter the Intersection Zone, their trajectory is locked in and they no longer make any adjustments. Figure 4.2 shows the structure of the intersection, with the Control Zone marked in green and the Intersection Zone marked in grey.

![Control Zone Diagram](image)

Figure 4.1: Intersection zones. Control Zone shown in green with Intersection Zone shown in grey.
4.3 Passing Order

The passing order defines what order the vehicles will traverse the intersection in and is ultimately what the MCTS solution is required to find. The passing order only comes into play when vehicles are on a collision course, in which case the vehicle with lower priority in the passing order will give way. This means that the passing order won’t necessarily define what order the vehicles traverse the intersection, vehicles may arrive out of order if they are not on a collision course.

4.4 Vehicle Adjustments

Collision courses are detected when two vehicles are on overlapping trajectories and are both scheduled to arrive at the intersection within a given window of time. When a collision course is detected, the vehicle with lower priority in the passing order will adjust its trajectory by slowing down so as to arrive at the intersection later than its initial projection, thus avoiding the collision. This adjusted trajectory means it may now be on a collision course with a different vehicle.

In order to calculate all adjustments required to avoid all collisions, each vehicle in the passing order ensures that it is not on a collision course with any vehicle that is higher priority than itself. This means that lower priority vehicles must take into account any adjustments that will be made by higher priority vehicles.

Once all the required adjustments are calculated, the vehicles execute the adjustments to their trajectories, thus arriving at the intersection in such a way that they have a collision-free route through the intersection.

4.5 Monte Carlo Tree Search

In our design, the passing order is modeled as a tree, with each node representing a vehicle and the hierarchy of the tree indicating the order that they will traverse the intersection, with child nodes arriving after their parent nodes.

In the selection step, the program will select the most urgently expandable node as governed by the tree policy. In the expansion step, a vehicle from the list of eligible vehicles is chosen and added as a child node to the selected node. In the simulation step, a full passing order is built out from the newly added child node and the cost for that passing order is calculated. Then in the backpropagation step, that cost is propagated through the selected nodes to update the value for each.

The computation budget for the MCTS solution is given in time with each vehicle given two
seconds to compute a passing order. This budget comes from the timing of the solutions which are calculated every two seconds. This does not factor in time for vehicles to exchange data, instead dedicating the entire two seconds to the MCTS computation.

4.5.1 Cost Function

The cost function is used to calculate the values of nodes during the simulation step of MCTS. It works by calculating the total amount of adjustment that would be required to facilitate the given passing order with no collisions. Whenever a vehicle is required to slow down, the adjustment total is increased by 1. Bearing in mind that when a leading vehicle slows down and 3 vehicles behind it will also need to slow down to maintain safe distance, the total cost of slowing down that leading vehicle will be 4.

The goal of our MCTS solution then is to find the passing order which results in the minimum total adjustment, i.e. the passing order that requires the least amount of vehicle slow down. During each simulation step, the total adjustment is calculated and backpropagated through each selected node. Since a traditional MCTS solution seeks to maximise the value of nodes, the cost function returns the total adjustment as a negative value so it can be maximised as normal.

4.5.2 Special Consideration

Our application of MCTS differs slightly from more typical applications like Go solutions. In a Go solution, the goal of one round of MCTS is to find the next move which maximises the computer’s chances of winning. When the budget is reached, that move is made and another round can be run in order to find the next move.

With our application, one round of MCTS must provide an entire passing order solution. It is not enough to know which vehicle should traverse the intersection next, the instructions for every vehicle will be needed. This problem was encountered by Xu et al. [2020] in their solution and we will be following their approach in our design.

Xu et al. [2020] solve this problem by making a slight modification to the MCTS methodology. As well as all the normal steps of MCTS, the solution must keep track of the best performing passing order encountered so far, known as the "state of the art passing order". As the tree is expanded and the partial passing order is built up, this state of the art passing order is continually replaced with better performing passing orders. When the computation budget is reached, the state of the art passing order is returned as the solution.
4.6 Decentralizing

In order to make our solution distributed, it is necessary to parallelize our MCTS solution. Leaf parallelization is used to improve simulation results and could be implemented in a distributed solution. This would require one agent to build the search tree while the other agents could weigh in on the parallel simulations. This introduces the problem of selecting a vehicle to build the tree.

Root and Tree parallelization offer a multi-agent approach where each vehicle can act as an independent agent contributing to the overall solution. Root parallelization allows each vehicle to act independently before unifying their results. Tree parallelization introduces the problem of maintaining a single tree among all vehicles and may also require one vehicle to be selected as the holder of the tree.

We decided to use the root parallelization approach, following the example of Golpayegani et al. [2015] who successfully used root parallelized MCTS in an electricity demand management application.

In the parallel solution, each vehicle acts as an agent and computes their own MCTS solution independent of the other vehicles. We then use the majority vote system put forward by Soejima et al. [2011]. When the passing order is determined, the vehicle votes for that passing order, broadcasting their vote to all other vehicles. Every vehicle receives all votes and implements the passing order with the most votes.

With no central control, each vehicle acts on the assumption that the other vehicles are implementing the same passing order when calculating adjustments. It is necessary then to have pre-programmed tie-breakers so that vehicles can be guaranteed to select the same passing order amongst all those put forward. The natural first tie-breaker to use will be the score of the passing order.

4.7 Heuristic

We will make one assumption in this solution which is used as a heuristic which informs the selection process during a simulation. This assumption is that it will never be more efficient for vehicles in the same lane to arrive out of order. Figure 4.2 illustrates this assumption. In the left pane, we see the vehicles that are eligible for selection in this iteration marked as yellow, with the green vehicle indicating the vehicle that was selected. In the right pane we see the next iteration. Again the vehicles that are eligible for selection are marked as yellow. We see that the green vehicle has been grayed out to indicate that it is already selected and the vehicle behind it is now eligible to be selected next.

In this way, only unselected vehicles that are leading their lane can be selected as the next
vehicle in the passing order. This streamlines the MCTS solution by drastically reducing the number of possible states and saving computational resources for more realistic scenarios, rather than simulating a passing order where one vehicle needs to overtake multiple others in the same lane which is highly unlikely to be the most efficient order.

![Figure 4.2: MCTS selection step. Yellow vehicles indicate eligibility for selection.](image)

### 4.8 Factors Affecting Performance

Since the computation budget for MCTS is given in time, the amount of time vehicles spend in the Control Zone has an effect on the quality of the solution. If the vehicles spend more time in the Control Zone, they can allocate more time towards computing a solution.

Factors that affect how long vehicles spend in the Control Zone include the size of the zone itself and the velocity of the vehicles as they pass through it. The size of the zone will be correlated to the range of communication that the vehicles are capable of, given that they communicate with each other within the Control Zone as mentioned in Section 4.2. If their communication range is 500m then it is obviously not possible to have a Control Zone with a 1km radius.

Similarly the speed limit of the road will dictate how long the vehicles spend in the zone. On slower roads vehicles will have more time available for computation. Thus the time delta determining the interval between solution computations mentioned in Section 4.1 can be fine tuned to the intersection based on these factors.
4.9 Security and Privacy Considerations

Sharing of location information amongst vehicles presents a privacy concern. A distributed solution provides a layer of protection by removing the single point of access at the intersection. However, if a bad actor were able to listen for broadcasts at intersections to intercept the vehicle communications, a vehicle’s entire route could be inferred by the intersections they visited and the paths they took through the intersections. To mitigate this concern, vehicles could use a randomised unique identifier at each intersection for the purposes of computing the solution. Once the intersection is traversed, the identifier is thrown out and a new one is generated at the next intersection. This prevents bad actors from collecting data for one vehicle from multiple intersections as they cannot correlate the data from the different intersections. This issue could be further mitigated by encrypting transmissions between vehicles, which would add another layer of protection to the system even if data is intercepted, although this would add to the computational overhead of the solution.

Another issue that could arise in the solution is the possibility of vehicles abusing the system to gain preference in the solution. One way vehicles could do this would be by spoofing multiple signals to make it appear as though there are multiple vehicles following behind it. This would increase the cost of slowing that vehicle and thus give it a higher priority in MCTS. This situation could be avoided by implementing a certificate system with each vehicle being issued a certificate which they use when exchanging data and each certificate being limited to one vehicle per intersection. This system would also aid in the inclusion of emergency vehicles with higher priority. Special certificates could be issued to these emergency vehicles with normal traffic being unable to spoof them for higher priority.

Malik and Sun [2020] go into further detail on possible cyber attack vectors against CAVs. They give an overview on attacks that were successfully made against self-driving Tesla vehicles by targeting the WiFi protocol and also the Google Assistant software used in those vehicles. They give a further list of possible attack vectors as well as a rating on the likelihood of such attacks being successful as well as the potential impact of a successful attack.

4.10 Summary

This chapter provided a full description of our design for a distributed, automated intersection management system. We began with a high-level overview of the design we had in mind. We then began by describing how we apply zoning to the intersection in order to define the areas in which the vehicles take certain actions. We discussed the idea of a passing order and how this is used by the vehicles to resolve conflicts and avoid collisions by
establishing a clear priority relationship between conflicting vehicles.

We then described how we integrate Monte Carlo Tree Search (MCTS) into our solution in order to compute optimal passing orders. Following this we described our method for parallelizing MCTS so that we can remove the central controller element and achieve a distributed system. We also described the heuristic that we utilise in order to streamline the solution and reduce the total complexity of the problem by minimising the number of possible passing orders allowed.

Finally we examined some of the factors that affect the quality of the solution, with time being the principal factor and this being limited by the capabilities of the vehicles as well as the desired velocity of the vehicles. We rounded out our design with a look at some possible security and privacy concerns that may arise within our system.
5 Implementation

This Chapter details our implementation of the design outlined in Chapter 4. Section 5.1 will give an overview of the software used in the implementation. Section 5.2 will then give a description of how the implementation was created using Python.

5.1 Software Overview

5.1.1 SUMO

Simulation of Urban Mobility or SUMO [2021] is a software package created by the Institute of Transportation Systems at the German Aerospace Centre which can be used to model traffic networks in urban environments. SUMO allows users to define a road network with designated routes through the network. SUMO then adds vehicles to the network, assigning each one a specific route and then modeling their passage through the network. SUMO comes with a Graphical User Interface to show vehicle traffic during the simulation.

SUMO has a wide array of functions to support a variety of applications. It has a built-in traffic light system for managing intersections with customisable timing of lights. Right-of-way rules can be programmed into intersections while the routes are used to define what turns are legal at an intersection.

It can support multiple types of traffic such as buses and emergency vehicles as well as simulating pedestrian traffic. SUMO also has a variety of modes of traffic control. Vehicles have different safety features which can be toggled on and off, such as keeping to speed limits, maintaining safe distances from other vehicles, obeying traffic lights etc. There is also an impatience mechanic which simulates driver impatience if they are forced to wait a long time at an intersection and become more likely to enter the intersection in an unsafe manner.

SUMO also comes with a variety of output options. Vehicle trip summaries can be output which give information on vehicle travel times, waiting times at intersections, distance traveled etc. SUMO is able to monitor for traffic collisions and output a list of the collisions that occurred during a simulation, including the vehicles that were involved and the location
of each collision. As well as collisions, it also notes dangerous decelerations, where a vehicle had to brake hard to avoid a collision.

In order to start a SUMO simulation, a SUMO configuration file must be defined. This file contains a reference to the network file to use in the simulation as well as the routes file which defines the routes vehicles can take through that network.

![SUMO Interface](image)

Figure 5.1: SUMO Interface.

5.1.2 Netedit

Networks and routes are defined in XML files. SUMO’s Network Editor tool Netedit [2021] provides a Graphical User Interface for defining networks and routes which can then be exported to XML files. Edges can be drawn with the mouse to quickly create a network which can then be fine-tuned by editing the position and length values. Each edge has a specific identifier and supports a customisable number of lanes. Overlapping edges result in an intersection which has its own identifier. Rules for the intersection can also be specified, such as right-of-way rules.

Netedit provides a fast way to get a network up and running as opposed to writing XML files from scratch. In our experience it was easiest to use Netedit to start the file and then fine adjustments such as changes to edge lengths could be made directly in the XML file via a text editor.
5.1.3 TraCl

Traffic Control Interface or TraCl [2021] is another tool that comes bundled with the SUMO package. The purpose of TraCl is to give a user access to a running SUMO simulation, allowing them to access values from the simulation such as vehicle locations and velocities. The user can also modify the simulation conditionally, giving instructions to specific vehicles to alter their velocity or route or altering things like traffic light states.

TraCl works by starting up a SUMO server with a given configuration file. TraCl then controls the simulation, progressing it in steps with all logic being performed in between steps and instructions being issued if needed. Users can manually step through the simulation in the SUMO GUI or set a delay between steps for the simulation to run automatically.

5.1.4 Python TraCl

In order to implement our solution, the Python TraCl [2021] library was used. This library acts as a wrapper around TraCl and allows a user to start up a SUMO simulation and control it with TraCl from a single Python script. Once the network file, route file and SUMO configuration file are all created, everything can be done in Python which streamlines the process of building and testing the implementation.
5.2 Python Implementation

As mentioned, our solution is fully implemented in Python. This section will give a rundown of how the different parts of the design were implemented. Section 5.2.1 will describe the SUMO network and route files that were created for the simulation. Section 5.2.2 describes the vehicle class which is responsible for managing vehicles in the simulation. Section 5.2.3 then describes the passing order class which is used to define the passing order for the vehicles as well as calculating the adjustments needed to meet this passing order.

Section 5.2.4 then describes how we implemented Monte Carlo Tree Search in order to determine an optimal passing order. Section ?? details the reporting class which is used to output metrics after a simulation is completed.

The main Python class is responsible for loading the SUMO configuration file and starting the simulation. This class will also add vehicles into the simulation using TraCI and then enter into a loop in order to step through the simulation. A list of added vehicles is maintained and used in order for the vehicle functionality to take place. After each step, each vehicle checks its current state and whether that state needs to be updated. They will also check if they are required to slow down according to the passing order and will initiate that adjustment if so.

A global configuration YAML file is used to control the parameters of the simulation. Examples of these parameters include:

- Passing Order Mode - Whether the passing order is first-come first-served or to be computed using MCTS.
- MCTS Time - The time budget for MCTS, usually configured to be two seconds but adjustable for the sake of experimentation.
- Number of Vehicles - The number of vehicles to be added into the simulation, used to test the solution in varying levels of traffic.

5.2.1 SUMO Network

A network is defined in which to add vehicles for the simulation. The network, seen in Figure 5.1, consists of four edges, each of which have two lanes which have opposed directions for traffic. Each edge is 1km long and all edges are given equal priority with no right-of-way defined at the intersection.

There are a total of 8 routes through the network. Vehicles on each edge can either go straight through the intersection or turn right.

The vehicle safety mode is set to 32 which turns off all safety features provided by SUMO.
This means the vehicles will not brake to avoid collisions and will not wait to enter an intersection in a safe manner. This safety mode ensures that when vehicles avoid collisions it is purely by virtue of our solution and not because SUMO intervened in order to avoid a collision. If the adjustments worked out by the solution are incorrect then the vehicles will collide.

5.2.2 Vehicle Class

The Vehicle class is used to manage all vehicles that are added to the simulation. Each Vehicle object has a number of associated attributes.

- veh_id - Used as an identifier for TraCI so each specific vehicle can be targeted.
- route - The route that the vehicle has been randomly assigned to traverse the network.
- state - The current state of the vehicle.
- veh_data - The data it has gathered from other vehicles during the information exchange phase of the solution. Vehicle data consists of a set of entries that correspond to another vehicle ID. Each entry has its own subset of attributes:
  - route - The route of the corresponding vehicle.
  - edge - The current edge that the corresponding vehicle is on.
  - distance - The distance of the corresponding vehicle from the intersection.
  - adjustment - The total adjustment that the corresponding vehicle needs to make.

The Vehicle class also provides various methods to support the solution.

- set_vehicle_state() - This function updates the vehicle’s state to the passed in state and performs whatever functionality is associated with the new state.

- is_outbound() - This function returns a boolean indicating whether or not the vehicle is on its outbound edge. Vehicles that have traversed the intersection already are considered outbound and are no longer a factor in the solution, even though they may be within the Control Zone.

- get_dist_to_intersection() - This function calculates the distance to the intersection. Ordinarily it is preferable to work with arrival time rather than distance, however time is difficult to work with in SUMO as the rate of time varies depending on the simulation settings. Since the vehicles travel at a fixed velocity, distance to the intersection becomes analogous to time to the intersection.
• **gather_veh_data()** - This function is what triggers the vehicles to exchange information with one another. Since there is no actual broadcasting of information, the list of all vehicles in the simulation is instead passed to each vehicle. They iterate through the list, searching for the ones that are in the Control Zone and use TraCI to retrieve the data required for each vehicle.

• **get_passing_order()** - This function triggers the computation of the passing order which is handled in another class. This function will either initiate a first-come first-served solution or a Monte Carlo Tree Search solution depending on the configuration.

### Vehicle States

Vehicles have a number of different states which affect their behaviour. Vehicles are colour coded according to their state so they can be easily identified in the simulator interface as shown in Figure 5.3.

- **Default** - Coloured grey, vehicles in the default state are not involved in the solution and should drive at a constant speed with no adjustments. Vehicles that are outside of the Control Zone or on their outbound edge are in the default state.

- **Control** - Coloured green, vehicles in the control state are within the Control Zone and not yet in the Intersection Zone. Vehicles in the control state gather information from all other vehicles in the control state and compute a passing order solution. Vehicles in the control state will turn orange to indicate that they are executing an adjustment to their trajectory according to the passing order and return to green when they have completed their adjustment.

- **Intersection** - Coloured white, vehicles enter this state when they arrive at the Intersection Zone. Vehicles in this state no longer adjust their velocity and are locked in to their current trajectory. They traverse the intersection in a uniform manner and transition to the default state when they exit the intersection.

#### 5.2.3 Passing Order Class

This class is used control the passing order for the intersection as well as calculating the adjustments necessary for vehicles to meet that passing order. A Passing Order object has the following attributes:

- **passing_order** - An array of vehicle IDs indicating the order by which they should traverse the intersection.

- **adjusted_order** - A dictionary which indicates the adjustment required by each
vehicle in order to avoid collisions at the intersection. Each vehicle ID is a key in the
dictionary with the value indicating the adjustment required for that vehicle. If a
collision is detected, a vehicle’s adjustment goes up by 1, which is equivalent to a 25
metre offset.

- **total_adjustment** - The sum of all required vehicle adjustments, used in evaluating
  a particular passing order.

The Passing Order class also has methods to facilitate the computation of
adjustments:

- **calculate_adjustments** - Calculates the adjustment required for each vehicle. If a
  vehicle is required to make an adjustments, this adjustment is also propagated to any
  vehicles that are following that vehicle in order to maintain safe distances and avoid
  collisions.

- **adjustment_required** - Helper function which returns a boolean indicating whether
  a given vehicle is on a collision course with any other vehicle. As per our design,
  vehicles only check whether they are on a collision course with vehicles that are higher
  than them in the passing order since they will never have to give way to vehicles that
  are lower priority. Vehicles that are arriving later in the order factor in the adjustments
  that the vehicles ahead of them will be required to make when checking for potential
  collisions.

### 5.2.4 Monte Carlo Tree Search

Our MCTS solution is based on the Python library provided by Sinclair [2021]. This library
provides a framework for computing a MCTS solution using a customisable State object.
This State object must implement the following functions in order for the solution to be calculated:

- `get_possible_actions` - This function is used in the MCTS expansion step in order to select a new child node to add to the tree. In our case, the nodes represent vehicles so the list of possible actions is the list of vehicles not yet selected. Our heuristic is also implemented here by restricting the list of possible vehicles to only those vehicles that are leading their lane.

- `take_action` - This function updates the partial passing order with the newly selected vehicle as well as removing that vehicle from contention for selection.

- `is_terminal` - This function is used to identify terminal states in the MCTS solution. In our case, a state is terminal when there are no more vehicles available for selection.

- `get_reward` - This is used as the cost function to calculate the value of nodes in the tree. We use the total adjustment as calculated by the Passing Order class as the reward function here. As per our design, the total adjustment is inverted to a negative so that the reward can be maximised as normal rather than changing the workings of the MCTS implementation.

### Special Consideration

As per our design, our implementation of MCTS needs to be slightly altered in order that a full passing order can always be retrieved when the time budget is reached. This is done by modifying the `execute_round` function of the Python library. Before the backpropagation step, the function also checks whether the passing order it just simulated is the best performing one it has checked so far. If so, it keeps a record of that passing order before backpropagating the results as normal. When the time budget is reached, the best performing passing order is returned rather than the node with the highest value.

### 5.2.5 Reporting

The final part of the implementation is a small reporting script that outputs a variety of metrics based on the simulation that was run. These metrics include:

- **Number of Collisions** - This is used to verify that the solution computed is successful at avoiding any collisions during the simulation. If this value is anything other than zero then the solution is invalid.

- **Total Adjustment** - The score of the selected passing order reflected by the total number of adjustments that the vehicles were required to make.

- **Average Duration** - The average time it took each vehicle to travel through the
network as provided through the SUMO output file. Since each of the four edges are of equal length, the randomisation of routes assigned to the vehicles has no bearing on the duration since all routes are of equal distance. Therefore the only factor affecting the duration is the need for vehicles to slow down to avoid collisions.

- Consensus - The number of votes that the selected passing order received from the vehicles. This is used to indicate the strength of the solution based on how many vehicles came to the same conclusion.

5.3 Summary

In this chapter we gave an overview of the software that we utilised in our solution. We described the network that we designed in order to simulate vehicle traffic. We then went into detail as to how we implemented our design in python, with a vehicle class designed to handle vehicle behaviour in the simulation and a passing order class used to enforce a passing order at the intersection.

We then describe the implementation of Monte Carlo Tree Search (MCTS), giving detail as to how the cost function is implemented as well as how MCTS iterates to a terminal state. Finally we described the reporting functions used in conjunction with SUMO’s output options in order to output results from a simulation.
6 Evaluation

To evaluate our solution, we ran multiple simulations under a variety of conditions. Alongside our solution we also test a first-come first-served policy, which is evaluated by simply taking the order the cars are already in with respect to distance from the intersection and using that as the passing order. The passing order class then calculates the adjustments in the same manner that it does for the MCTS solution.

We also ran a traffic light system (TLS) to compare against a traditional signalized intersection. The TLS is managed by SUMO using the default settings for a traffic light intersection. Lights stay green in one direction for 42 seconds, then orange for 3 seconds before turning red. Then the opposing direction goes through the same cycle for a total of a 90 second cycle. The SUMO safety features are turned back on for this simulation so that the vehicles will obey the lights.

The metrics we look at are as follows:

- **Adjust** - The total adjustment required by all vehicles averaged across the ten simulations. This is given for the MCTS and the FCFS systems. There is no value for the TLS system since we don’t calculate any adjustments in that system.

- **Standard Deviation $\sigma$** - The standard deviation of the Adjust figure across the ten simulations for each vehicle density. This lets us examine how consistent the solutions found by MCTS and FCFS are.

- **Time (s)** - The average total time taken by the vehicles in traversing the network from start to finish, given for the MCTS, FCFS and TLS systems.

- **Consensus** - The total votes that the selected passing order received in the MCTS solution.

To generate results, a simulation for each policy was run ten times with the average values and the deviation calculated from the results of the ten simulations. These ten simulations were repeated in three different scenarios with differing numbers of vehicles added into the simulation to examine how increasing traffic density affects the performance of each solution.
The results of all these simulations can be seen in Table 6.1. We can see that both the MCTS and FCFS considerably outperform SUMO’s TLS implementation. This is to be expected as both solutions allow for zero vehicle stoppages at the intersection while the TLS will almost always require some stoppages.

Figure 6.1 illustrates the performance of each solution with respect to time at the varying levels of traffic density. Figure 6.2 compares the total required adjustment for the MCTS and FCFS at each level of traffic density while Figure 6.3 compares the standard deviation of the adjustment levels across the ten simulations for MCTS and FCFS.

We can also see that the MCTS solution is able to outperform the FCFS solution in each of the three categories. Both the total required adjustment and the average journey time are lower and the MCTS solution also has a lower standard deviation, indicating that it offers more consistent results.

![Graph comparing total time of MCTS, FCFS and TLS.](image)

**Figure 6.1: Graph comparing total time of MCTS, FCFS and TLS.**

**Table 6.1: Evaluation results**

<table>
<thead>
<tr>
<th></th>
<th>Adjust</th>
<th>$\sigma$</th>
<th>Time (s)</th>
<th>Adjust</th>
<th>$\sigma$</th>
<th>Time (s)</th>
<th>Consensus</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Veh</td>
<td>14.5</td>
<td>3.29</td>
<td>147.19</td>
<td>8.4</td>
<td>3.69</td>
<td>146.29</td>
<td>1.3</td>
<td>170.48</td>
</tr>
<tr>
<td>30 Veh</td>
<td>125.1</td>
<td>23.7</td>
<td>151.06</td>
<td>78.1</td>
<td>9.95</td>
<td>148.86</td>
<td>1.0</td>
<td>180.01</td>
</tr>
<tr>
<td>50 Veh</td>
<td>374.2</td>
<td>43.27</td>
<td>155.78</td>
<td>213.0</td>
<td>20.27</td>
<td>150.76</td>
<td>1.0</td>
<td>180.56</td>
</tr>
</tbody>
</table>
Discussion

The total adjustment is not necessarily correlated to the duration because it depends on how the adjustments are performed. If the adjustment algorithm is very inefficient then higher total adjustments will be punished more than may be accurate. This is why we look at both the adjustment and the time for our particular method of adjusting trajectories.

There is one caveat to add here. As mentioned, TraCI works by stepping the simulation forward by 1 step and then performing whatever logic is necessary depending on the conditions of the simulation. SUMO in turn calculates time based on these steps rather than the real passage of time. This effectively means that as far as the vehicles are concerned, all computation happens instantaneously with basically zero computation time. This may not have much bearing on the results however, as the idea is that the vehicles perform this computation while they are driving through the Control Zone and then execute the required adjustments to their trajectory. If the Control Zone is large enough to allow the vehicles the two seconds of MCTS computation as well as the communication overheads and the time to adjust their trajectory then the computation time will not factor into the journey time in any case.

Another major takeaway from the results in Table 6.1 is the very low level of consensus in the MCTS solution. For the simulations with 10 vehicles, three of the results had a
consensus of 2 while every other simulation had a consensus of just 1. The means that the majority of the time, what happens is one vehicle has its computed passing order selected as the overall passing order while the result of every other vehicle is thrown out. This strongly indicates that root parallel MCTS is not the way to go if we want to make efficient use of the parallel processing power available to us in a distributed solution.

While the MCTS solution was able to outperform the FCFS policy, it's fair to say that it is not that far ahead of it, particularly as we get into higher densities of traffic where the solution of Xu et al. [2020] seemed to vastly outperform a FCFS policy. The low level of consensus may be part of the problem. In a distributed solution, you have a phenomenon with higher traffic densities as the complexity of the problem increases but so too does the available processing power as more vehicles are available to join the solution. In our implementation, we are not seeing the benefit of the additional processing power as in our results, we only ever have one vehicle calculating the solution for the entire intersection.
7 Conclusion

The first conclusion to draw from this paper is that it is possible to use a parallel implementation of Monte Carlo Tree Search to achieve a distributed solution to traffic management of connected autonomous vehicles at unsignalized intersections. This MCTS solution is capable of vastly outperforming a traditional traffic light system and also capably of outperforming a FCFS policy.

The second conclusion to draw is that the root parallel method of MCTS with majority voting is not an efficient usage of the resources available as it is so rare for the vehicles to reach any consensus. If the time budget was larger or the vehicles had more computational power available to them, they may be capable of reaching some form of consensus. With the constraints as they are, they are unable to explore the search tree deeply enough that they start to come to the same conclusion. It could also be that there are multiple solutions available with similar efficiency so even with extra resources they would not necessarily find the same solutions. With time being a particularly precious resource at the intersection, it seems more prudent to investigate an alternate method of parallelizing rather than increasing the resources available.

7.1 Future Work

Due to the issue with the root parallel method for MCTS, the natural next step to investigate would be the tree parallel method. The tree parallel method works by having all agents collaborate on the same search tree, allowing for greater exploration of the tree than would be possible with a single agent. This would allow for much more efficient usage of the processing power of the vehicles as every vehicle would be contributing to the overall solution. In theory this should produce stronger results than were obtained using the root parallel method.

It would also be interesting to explore more complex intersections. Adding in extra lanes and allowing for more routes through the intersection would increase the complexity of the problem. Since MCTS excels in problems involving a large number of possible states, this added complexity should only improve the performance of the MCTS solution relative to the
FCFS policy.

We could also look at removing some of the assumptions that were made. Allowance for emergency vehicles could be made by altering the cost function to factor in a vehicle weight parameter. Alternatively if we want emergency vehicles to jump to the head of the queue, the start of the passing could be fixed to allow for that and the solution would compute the remainder of the order. We could also look at the impact of communication latency on the solution and how the vehicles would recover from packet loss. Finally we could investigate how our MCTS solution could be implemented in a mixed traffic scenario where there are human drivers as well as connected autonomous vehicles. In this case, the CAVs would have to compute a solution working around the limitations of human drivers in executing fine adjustments to their trajectory.
Bibliography


