Real-Time Intelligent Autonomous Intersection Management Using Reinforcement Learning

Udesh Gunarathna[1], Shanika Karunasekera[1], Renata Borovica-Gajic[1] and Egemen Tanin[1]

Abstract— **Autonomous intersection management has the ability to reduce congestion at intersections significantly, compared to classical traffic signal control in the era of connected autonomous vehicles. Autonomous intersection management requires time and speed adjustment for vehicles arriving at an intersection for collision-free passing through the intersection. Due to its computational complexity, this problem has been studied only when vehicle arrival times towards the vicinity of the intersection are known beforehand or with other simplifying scenarios which limits the applicability of these solutions for real-time settings. To solve the real-time autonomous traffic intersection management problem, we propose a reinforcement learning (RL) based multiagent architecture and a novel RL algorithm coined multi-discount Q-learning. In multi-discount Q-learning, we introduce a simple yet effective way to solve a Markov Decision Process by preserving both short-term and long-term goals, which is crucial for collision-free speed control. Our experimental results using microscopic simulations show that our RL-based multiagent solution can achieve near-optimal performance efficiently when minimizing the travel time through an intersection.**

I. Introduction

The emergence of Connected Autonomous Vehicles (CAVs) is expected to revolutionize traditional traffic management solutions. In future traffic management, both CAVs and the traffic infrastructure can act as intelligent agents who work hand in hand to reduce traffic congestion in real-time [1], [2]. The traffic infrastructure (i.e. a traffic controlling agent) can optimize the road network parameters based on the information from CAVs, and CAVs can optimize their driving behaviors according to the traffic controller's instructions. In particular, traditional traffic management solutions at intersections can be significantly improved, as congestion at intersections is critical for the overall performance of a road network and therefore, solutions on this front are vital [3]. Autonomous Intersection Management (AIM), where the intersection is controlled through appropriately scheduling vehicles for collision free passing though the intersection has been proposed as a solution. In this work, we propose a reinforcement learning based multiagent solution for AIM which allows vehicles to manage their speed and arrival time at the intersection to reduce congestion at intersections.

AIM is a paradigm proposed by Dresner *et al.* [4] to replace the traditional traffic signal control. In AIM, each CAV arriving towards an intersection reserves a time to traverse the intersection crossing point via an intersection manager. Then, each CAV's speed is controlled to adhere to the schedule while guaranteeing safety. Due to the ability of AIM to reduce the congestion at intersections [5], AIM has been widely studied [6], [7]. However, most problem formulations assume that the arrival times of vehicles to the vicinity of an intersection are known *a priori*. This assumption does not hold when dealing with real-time traffic, which limits the applicability of these solutions to real scenarios [8].

For AIM to be applicable to real-life traffic control, the speed of the vehicles has to be computed in real-time, and the computational time for this plays a critical role in the feasibility of AIM. Previous efforts exhibit high computational time [9] as they rely on mathematical programming or analytical methods. The impact of high computational time is two fold. First, suppose the intersection controller takes a long time to compute scheduled times. In that case, the positional difference of CAVs before and after the computations, poses a significant safety risk of the vehicle crashing due to the given scheduled times not being feasible anymore. Second, if the time for computing CAV speed for a trajectory is high, the remaining time after the computation may not be sufficient to reach the intersection crossing point precisely at the scheduled time.

A recent work [8] develops a stochastic solution to the problem, assuming vehicle arrival times are not known beforehand. However, their solution employs linear programming (LP) for every CAV's arrival computation, which is prohibitively expensive, as demonstrated in Section VIII. Further, their method is only applicable to intersections with single lane roads without turning directions.

Our work fills the research gap by providing a computationally efficient, intelligent, multiagent solution based on Reinforcement Learning (RL) for AIM. Our solution consists of two main sets of agents. The first type of agent is a polling-based coordinating agent (intersection controller) positioned at the intersection. The second component is a set of distributed RL agents, which are assigned on a per vehicle basis. The coordinating agent communicates with the RL agents to schedule time intervals to reach the intersection for each CAV that is within a certain distance from the intersection. The coordinating agent uses a novel polling algorithm to handle multi-lane intersections with multiple turning directions, overcoming thereby a major limitation of previous work where only intersections with no turning directions and single-lane roads were possible [8]. Once the coordinating agent provides a time schedule, an RL agent controls each vehicle's speed to adhere to the coordinating agent's time schedule. The advantage of such an approach

[1] School of Computing and Information Systems, The University of Melbourne, Australia

is that once an RL agent is trained offline, decision-making can be done much faster online. This avoids the computation overhead incurred by other techniques.

The learning task for the RL agent is two fold: (1) learn to control a vehicle's trajectory to reach the intersection precisely at the scheduled time, and (2) keep a safe distance from the vehicle in front. Keeping a safe distance is a task with a short-term goal, because the front vehicle can change its driving behaviour (the driving behavior of an RL agent or a human) in short time-intervals. In contrast, reaching the intersection at a scheduled time requires long-term planning because successfully reaching the intersection is only determined at the end of the trajectory. Combining such two learning problems into a single task, as shown in previous work [10] will only learn one of the problems, because each learning problem contains an objective with a different time-horizon. Existing RL algorithms such as Q-learning use a fixed parameter named *discount factor* to control the length of the time-horizon. Using a fixed discount factor focuses on learning either the short-term or long-term task successfully [11], and fails when both short-term and long-term tasks need to be learned simultaneously, hence being unsuitable for our task. We propose a novel RL algorithm coined *multi-discount Q-learning* to achieve short-term goals, while following a long-term goal in a single Markov Decision Process. We believe our proposed method is applicable to other problem-domains that exhibit a mix of long-term and short-term goals, such as robotics [12].

Our contributions are four-fold: (1) We propose a computationally efficient multiagent solution for AIM. (2) We introduce a novel reinforcement learning algorithm that can effectively learn multiple learning tasks at the same time. (3) We propose a novel polling algorithm to handle multi-lane intersections with multiple turning directions. (4) We demonstrate the effectiveness and efficiency of our approach against several baselines using microscopic traffic simulations.

II. Related Work

A. Autonomous Intersection Management

There are two inter-dependent sub-problems that have been studied in the literature to optimize the AIM; (1) find a distinct time-schedule for each incoming vehicle to arrive at the intersection, and (2) find a vehicle trajectory such that a vehicle arrives at the intersection exactly at the schedule time[1]. The existing work can be divided into two main categories based on the proposed solutions to the aforementioned two problems.

The first category of work optimizes the time-schedule (sub-problem (1)) as a scheduling problem and then computes a sub-optimal vehicle trajectories which adhere to the optimized time-schedule [7], [14]–[17]. The optimal scheduling optimization is NP-hard [9], [18], which makes it computationally expensive. Another drawback of this kind of approach is that all the arrival times of vehicles to the vicinity of the intersection need to be known *beforehand* to optimize the time-schedule.

The second category of work focuses on finding a safe trajectory or fastest trajectory as a solution to sub-problem (2), whilst employing a heuristic to compute the time-schedule to sub-problem (1) [6], [19], [20]. Finding the optimal trajectory is however prohibitively expensive in real-time when using a method like linear programming, as demonstrated in our experiments. For example, Au *et al.* [20] uses an analytical method to find the trajectory using a set-point schedule and a bisection method. However, to simplify the search space when deciding on the trajectory their work does not consider the maximum velocity, which leads to a sub-optimal trajectory. Even though these approaches are able to compute trajectories, they do not optimize sub-problem (1). Thus, they do not optimize the throughput nor reduce waiting time at intersections. In contrast, our objective is to maximize the throughput at the intersection.

Recent work [8] proposed a solution to optimize the throughput in a stochastic setting using a polling system and linear programming. The polling system is used to optimize the time-schedule for each vehicle. Then, a linear program is solved for each vehicle to find its trajectory. These linear programs need to be computed sequentially (centralized). This means that the linear program for the first vehicle arriving at the intersection should be computed first, and then the next vehicle, and so on. Although this approach can successfully solve the stochastic AIM problem, computational time required for linear programming hinders the applicability of this solution for real-time usage. On the contrary, we propose a distributed learning-based solution, which enables real-time AIM.

B. RL with Variable Discounting

Learning a task is difficult when there are objectives with different time scales. The time scale of a task is directly impacted by the discount factor of RL agents (e.g. Q-learning or SARSA) [11]. Thus, most past efforts focus on changing the discount to learn tasks with multiple time scales. Edwards *et al.* [21] extends the LP formulations of [11] and propose a multiple discount SARSA algorithm by considering the reward as a vector and using a separate action-value function (Q-function) for each sub-task. Human intervention is then needed to find the best policy among these sub-tasks. Burda *et al.* [22] follows a similar approach to combine intrinsic and extrinsic rewards. An automated approach is proposed by Li *et al.* [23] to combine different objectives learned by a set of factored Q-functions using a lexicographic ordering of objectives. Finding such lexicographic ordering of objectives is non-trivial and can be problem-dependent. In the above-mentioned approaches, each sub-task is learned separately. Because of that, the inter-relationship between sub-tasks is ignored, and the number of parameters to be learned is high. In contrast, we propose a simple and memory-efficient method to learn each sub-task using a single action-value function (a single Q-function) whilst preserving the time scales of sub-tasks. As we show in our experiments,

[1] If the objective is to optimize the throughput then the vehicle trajectory should reach the intersection at the maximum speed as well [13].

our proposed method is able to achieve superior results in achieving both short-term and long-term goals.

III. Background

Our proposed architecture uses a polling system to schedule the incoming CAVs, and uses Q-learning to determine the CAV trajectories. We provide a brief introduction to both.

Polling system: A polling system consists of a single server and a set of queues. Each queue contains a number of customers (in First-In First-Out (FIFO) order). Customers may arrive at the queue in a stochastic order. The server can start serving a customer from any queue. The term *service time* is the time taken to service one customer. Once the server has serviced the first customer, the server can select the next customer from any queue. When switching between queues, the server has to wait for an additional time called *switch over time*. The strategy that the server uses to determine from which queue the next customer is selected is called the *policy*. There are several policies in the literature such as *K-limited*, *gated* and *exhaustive*. The definitions of customers, *switch over time* and *service time* related to AIM are described in Section V-A.

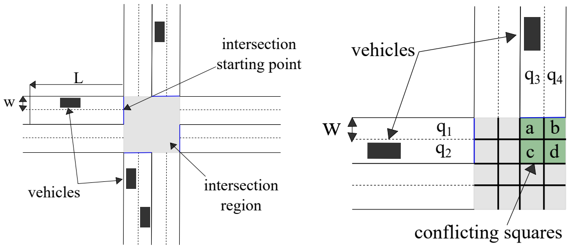
Q-learning: In RL, a problem first needs to be formulated as a Markov-decision process (MDP). An MDP consists of a state space $\mathcal{S}$ and an action space $\mathcal{A}$. When an action $a_t \in \mathcal{A}$ is taken in the current state $s_t \in \mathcal{S}$, at time t, the MDP's state changes to s_{t+1} according to the transition probability $\mathcal{T}(s_t, a_t, s_{t+1}) = Pr(s_{t+1}|s_t, a_t)$. The MDP provides a reward r_t for the transition where r_t is assigned according to $\mathcal{R}(s_t, a_t, s_{t+1})$. An RL agent acting on the MDP consists of a policy $\pi(a|s)$ which describes the agent's behaviour. The policy $\pi(a|s)$ indicates the probability of an agent taking the action a in the state s. The objective of the agent is to maximize the *expected reward* G_t starting from any given time step t. The expected reward is defined as $G_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau$, where t is the current time step, γ is the discount factor and T is the time that MDP reaches its terminal state.

The action-value function (Q-function) for policy π stores the expected reward by taking the action a in the state s defined as: $Q^\pi(s,a) = \mathbb{E}[G_t|s_t = s, a_t = a, \pi]$. Q-learning approximates the optimal Q-function iteratively by observing the transitions (s_t, a_t, s_{t+1}, r_t) at every time step when $\mathcal{T}$ is unknown. Considering the reward from n number of steps we get the following n-step Q-learning equation.

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[\sum_{\tau=0}^{n-1} \gamma^\tau r_{t+\tau} + \gamma^n \max_{a'} Q(s_{t+n}, a') - Q(s_t,a_t)] \quad (1)$$

Deep Q-learning: Deep Q-learning (DQN) [24] uses a neural network to approximate the tabular Q-function when the number of state-actions is large. The deep neural network is represented as $Q(s,a;\theta)$ where θ is the set of parameters of the neural network. DQN learns the Q-function minimising the following loss function w.r.t θ.

$$L_t(\theta_t) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}[(y_t^{DQN} - Q(s,a;\theta_t))^2] \quad (2)$$



(a) A four legged intersection with vehicles represented with black rectangles. The control region for one road segment is shown with length L. The intersection region is shown in gray and the intersection starting point for road segments are shown in blue lines.

(b) The figure shows two 2-lane roads segments where each lane is represented as a queue. The green points show four possible conflicting squares where two vehicles from 2 lanes could collide, if vehicles coming from q_1 to q_4 are going straight through the intersection.

Fig. 1: Multi-lane signal-less intersection

$$y_t^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta_t^-) \quad (3)$$

where y_t^{DQN} is the *target function* and θ^- parameters represent a separate neural network that keeps its parameters frozen for a number of time steps while (θ) parameters are optimized. After a certain number of time steps θ parameters are copied to θ^-.

IV. Problem Definition

Let us assume that an intersection I is connected with K road segments. A vehicle i travelling towards the intersection from road segment $k \in K$ is defined as $v_{i,k}$. A vehicle i consists of dimensions $l_{v_{i,k}}, w_{v_{i,k}}$ where $l_{v_{i,k}}, w_{v_{i,k}}$ are the length and width of the vehicle respectively. An example of a four legged intersection and vehicles arriving towards the intersection are depicted in Figure 1a.

Intersection region and starting point: Intersection region is defined as the common intersecting area of all road segments at the intersection (I). We define the point where a road meets the intersection region as the *intersection starting point*. The intersection region and intersection starting point are shown Figure 1a.

Control region: The control region of a road segment k is a road segment of length L measured from the intersection starting point along road segment k, where L is defined as the control region length. The control region of the intersection consists of control regions of all road segments.

Vehicle position: The position of vehicle $v_{i,k}$, $x_{v_{i,k}}(t)$ is the distance from the intersection starting point to the vehicle's front bumper at time t.

Safety: Safety at the intersection and control regions at time t is achieved when any two vehicles' occupied areas do not have overlap, where the occupied area of a vehicle is a 2-d space covered by $l_{v,i} \times w_{v,i}$ dimensions of the vehicle w.r.t. its current position.

Travel time: The travel time of a vehicle $v_{i,k}$, $TT_{v_{i,k}}$ is defined as the time taken for the vehicle to cross the intersection, i.e. the time between when the vehicle enters and exits the control region.

Total travel cost: Given that there are N_t vehicles inside the control region at time t, the total travel cost $TTC(t)$ is the sum of the expected travel times of all N_t vehicles.

Problem Statement: Given that the vehicle arrival times to control region are stochastic (the vehicle arrival times are not *known a priori*), our objective is to minimize the $TTC(t)$ at each time step while guaranteeing safety at every time step.

V. Real-time Intelligent Autonomous Intersection Management

A solution to the aforementioned optimization problem requires every vehicle in the control region to have a distinct scheduled time to reach the intersection starting point, and a well-defined trajectory to reach the intersection starting point at the scheduled time. We can reduce the computational complexity of the overall optimization problem by decoupling these two optimizations as two separate sets of agents, which can act cooperatively. Two optimization problems are (a) a scheduling optimization problem and (b) a trajectory optimization problem. Thus, we propose an intelligent multi-agent solution named **Coordinated Multi-discount Q-learning for Autonomous Intersection Management (CMQ-AIM)** which consists of two main components. The first component is a *Polling-based Coordinating* Agent that schedules a time to arrive at the intersection region for every vehicle in the control region (scheduling optimization). The second component is a set of distributed *RL-based Trajectory Control Agents* (we will use the name RL-based Agent for short) each of which controls the trajectory of a vehicle so that the vehicle reaches the intersection region precisely at the scheduled time given by the coordinating agent at the maximum speed (trajectory optimization) [13]. In the next sections we describe the two types of agents in detail.

A. Polling-based Coordinating Agent

The Polling-based Coordinating Agent uses a polling algorithm and communicates with the RL-based Agent to send/receive relevant information. We first show how a vehicle's arrival can be modelled as a polling system. We then formulate the polling system for an intersection with single-lane roads, similar to [8]. Finally, we extend the formulation to a multi-lane intersection with multiple turnings.

Each incoming road segment can be represented as a queue, and elements in the queue (customers) are the vehicles within the control region. The *service time* should ensure that two vehicles from the same road will not reach the intersection simultaneously. Thus, the *service time* is set to the time duration between when the vehicle's front bumper enters the intersection region, and the vehicle's rear bumper enters the intersection region while travelling at the maximum speed. The *switch over time* should ensure that a vehicle will enter the intersection only when a vehicle from a different road segment has exited the intersection region. Thus, the *switch over time* is set to the time duration between when the vehicle's front bumper enters the intersection region, and its rear bumper exits the intersection region while travelling at the maximum speed.

Once the *service* and *switch over* times are defined, any regular polling policy [25] can be used to compute a schedule for every vehicle in the control region, and the time-schedules will be non-conflicting. The polling algorithm is event-triggered, implying that when a new vehicle enters the control region, a new schedule will be computed. Thus, the polling algorithm can handle stochastic vehicle arrivals.

The above formulation that represents an intersection as a polling system is capable of handling intersections with single-lane road segments only. When there are multiple lanes per road segment, there is more than one conflicting point (see Figure 1b). To address this limitation in a computationally efficient manner, we propose a Multi-Lane Polling System discussed next.

Multi-Lane Polling System: The proposed system represents each lane in the intersection as a separate FIFO queue, as shown in Figure 1b. Each queue contains vehicles that have not yet finalized the time schedule. We denote all queues as a set Q. With multiple conflict points between queues in Q, the time that a queue has to wait after processing another queue is not constant across queues compared to the single-lane scenario. Thus, instead of using a fixed service or switch over times we propose a queue dependent function named *queue transition function* $f_q : q_1 \times q_2 \mapsto \mathbb{R}$. The function represents the time that the polling system should wait to process a customer (vehicle) in q_2 after servicing a customer from q_1, i.e. it represents the switch over time between q_1 and q_2 plus the service time of q_1. For two consecutive requests from the same queue, the f_q output equals to the queue's service time. For two queues in the same road, f_q is equal to zero as queues can be processed in parallel. For the scenario shown in Figure 1b the queue transition function is a 4x4 matrix. Once the f_q is defined, the polling order can be computed using the following algorithm. With a given Q, the algorithm first initializes the set T_a which is used to store the latest scheduled time (of the vehicle) assigned to each queue [2]. In line 2, the first queue to process is selected. In line 4, the

Algorithm 1: Multi-Lane Polling Algorithm

```
  Input: $Q = \{q_1, ...q_N\}$:$N$ number of queues for
         each lane
  Input: $\tau$ : time to reach the intersection at max
         speed from control region length $L$
1 $T_a = \{t_1, ...t_N\}$ :
    latest service time of each queue initialized with zeros
2 $q \leftarrow$ select the queue with the first arrived vehicle
3 while queues in $Q$ are not empty do
4     $v \leftarrow q.pop()$
5     $t_v \leftarrow \tau + \max_{q_i}(T_a[q_i] + f_q(q, q_i))$ :
        ignore queues with $f_q(q, q_i))$ equals to zero
6     $T_a[q] \leftarrow t_v$
7     $q \leftarrow$ select the next queue to process
```

[2] The vehicles are allowed to change lanes (i.e. change the queue) inside the control region, until a vehicles receives a finalized time schedule.

algorithm takes the first vehicle v from the queue q and finds the latest safest schedule time t_v that can be assigned to v. Taking $\max_{q_i}(T_a[q_i] + f_s(q, q_i))$ ensures that the assigned time will not conflict with previously assigned times for all queues. Then, t_v is stored in T_n as the latest scheduled time for q. The next queue to process will be selected using the aforementioned polling policies (e.g. exhaustive, k-limited, or gated) in line 7. The only added computation over the standard polling algorithm is the max operation (linear in the number of queues).

Extending to all roads segments and multiple turning directions: For simplicity, we explained our Polling-based Coordinating Agent only with two road segments with vehicles moving straight through the intersection. Extending this setup to all road segments is trivial. As in Figure 1a, when we have eight incoming lanes connecting to the same intersection, the queue transition function then becomes an 8x8 matrix. Then, to handle right and left turns, let us assume that each lane can have two allowed directions (either straight and left turn or straight and right turn). Then, each lane can be represented by two distinct queues, and the total number of queues will be 16. Finally, the queue transition function (f_q) becomes a 16×16 matrix. We can generalize this matrix to $(n_l * 2) \times (n_l * 2)$, where n_l is the number of incoming lanes. This matrix then can be used in **Multi-Lane Polling Algorithm** (Line 5). Note that the matrix size does not increase exponentially in real-world intersections as most intersections consists of 4 or 5 incoming roads. The number of lanes per road segment are normally within the range of 2 to 5 incoming lanes.

B. RL-based Trajectory Control Agent

Once the Polling-based Coordinating Agent schedules a time for a vehicle to reach the intersection region, the next objective is to control the vehicle's speed so that the vehicle reaches the intersection region precisely at the scheduled time. A vehicle needs to reach the intersection region at maximum speed to ensure the *service* and the *switch over* times are not violated. The higher the speed the higher the throughput will be [26]. A naive way to achieve this arrival to the intersection at a given time and at maximum speed is by stopping at a considerable distance away from the intersection starting point for some time and then only accelerating at the last minute. This sort of trajectory will block vehicles that follow, which reduces the control region's capacity [8]. Thus, an optimum trajectory should stay as close as possible to the intersection. In previous work [8], this is achieved by solving a linear program for every vehicle starting from the leading vehicle in a road in a sequential order. However, solving a linear program every time the Polling-based Coordinating Agent updates the time schedules requires a significant amount of computation. If the trajectory is not computed fast enough, there is a risk of crashing with a former vehicle or not reaching the intersection at a scheduled time. Furthermore, LP-based methods require the complete trajectory of the front vehicle to be known when computing the current vehicle's trajectory. This information is however unavailable when the front vehicle is not a CAV. Even if the front vehicle is a CAV, a re-computation is needed if the front vehicle's trajectory is changed. To overcome these limitations, we propose a Deep RL-based distributed solution where an agent is trained to control the vehicle's speed to reach the intersection precisely at the scheduled time. Once an RL-based Agent is trained offline, it can be used to make real-time decisions. This approach mitigates the safety concerns posed by the computational overhead of LP.

We use the Q-learning algorithm to learn the vehicle's trajectory to reach the intersection region at a maximum speed (a.k.a. trajectory control). While learning to control the vehicles' trajectory to reach the intersection region, the agent should learn to keep a safe distance from the former vehicle (a.k.a. cruise control). See Section VI-A for the state, action, reward definitions. Combining these two objectives can be achieved by combining two reward functions from each task [10]. However, we observe that there is a unique property of this problem that makes a standard Q-learning algorithm unsuitable. In the trajectory control task, successfully reaching the intersection is determined only when the vehicle reaches the intersection starting point (i.e. at the end of an episode), making the objective long term. Whereas in the cruise control task, the former vehicle can be humanoid/autonomous, and its behavior can be changed in short time intervals, i.e. the vehicle behaviour is only deterministic in a short time interval. The existing Q-learning algorithm can be set either as a short-term or as a long-term objective task using the *discount factor*. Suppose the discount factor is set as a long-term task, the former vehicle's behavior cannot be adequately modeled and will result in a sub-optimal behavior (i.e. keeping a large distance from the former vehicle). If the discount factor is set as a short-term task, the RL-based Agent is unable to see a large number of time-steps ahead. Then the trajectory control task cannot be achieved properly because it requires controlling the vehicle a long distance away from the intersection. We propose a novel Q-learning algorithm coined **Multi-discount Q-learning** to learn the two tasks to preserve the short-term goal while following the long-term goal.

VI. Learning Long-term and Short-term Goals

This section first introduces a Multi-objective MDP (MO-MDP) and explains why the standard Q-learning cannot learn multiple tasks with different temporal objectives. MO-MDP is different from MDP only in the reward function definition. In MO-MDP the reward function is a vector $\mathbf{r_t} \in \mathbb{R}^k$ where k is the number of objectives/tasks. Learning with MO-MDP be considered for 2 main use cases, based on the information available about the MDP. The first scenario is when we know how much weight should be given to each objective (relative importance of each reward/task). That means we know a weight vector $\mathbf{w} \in \mathbb{R}^k$, so that we can compute $\mathbf{w}.\mathbf{r_t}$. The second scenario is when we do not know how much weight should be given to each objective while learning, i.e., $\mathbf{w}$ is unknown. The former is solved by transferring into a single-objective MDP, and the latter is solved by learning objectives separately and combining them at the decision-making stage [10].

Our problem consisting of the trajectory control and cruise control tasks belongs to the former case where we know the weight vector $\mathbf{w}$. However, in our case, each task contains a different temporal objective. Before introducing the proposed algorithm, let us discuss why traditional Q-learning is unsuitable for this problem. First, we can modify the standard Q-learning equation (Equation 1) to work with weight vector $\mathbf{w}$ in the MO-MDP defined above as follows.

$$Q(s_t, a) \leftarrow Q(s_t, a_t) + \alpha[\sum_{\tau=0}^{n-1} \gamma^{\tau}(\mathbf{w}.\mathbf{r_{t+\tau}}) + \gamma^n \max_{a'} Q(s_{t+n}, a') - Q(s_t, a_t)] \quad (4)$$

Note that in Equation 4, when combining k number of tasks, discount factor γ is a scalar value between 0-1 and it is a common value for all the elements in $\mathbf{r_t}$. The discount factor determines how much importance is given to future rewards at the current time step t. The lower values of discount factor (0-0.9) typically have a short-term effect because the amount of future rewards considered can be approximated as $1/(1-\gamma)$ [27]. For example, in the extreme case when γ is 0, we only consider the immediate reward, while with γ of 0.9 only the next 10 rewards have an impact. In both cases, the value of $\max_{a'} Q(s_{t+n}, a')$ and rewards from $t+1$ to $t+n$ have less impact to the current value. The objective we are trying to achieve becomes a short-term objective. When γ is 1, all the future values of reward have the same importance. In that case, the objective becomes a long-term objective. Since we can only assign a single scalar value to the discount factor γ, the combined task becomes either a long-term or a short-term objective task.

Multi-discount Q-learning: We introduce two components to overcome the aforementioned limitation. First, we introduce a discount vector $\mathbf{\Gamma} \in \mathbb{R}^k$ which can assign different discount factors to each objective for k objectives of Multi-objective MDP. The expected return defined in Section III can be modified as $G_t = \sum_{\tau=t}^{T} \sum_{i=0}^{k} \gamma_i^{\tau-t} r_{t+\tau,i}$, where $\gamma_i \in \mathbf{\Gamma}$ represents the discount factor assigned to the i^{th} objective and $r_{\tau,i}$ is the reward received for the i^{th} objective at time step $t+\tau$. Each reward is associated with its own discount factor rather than a common one for all rewards. Since each reward is discounted differently, considering the reward is long-term or short term, temporal length of each objective is preserved.

The second aspect considers how to discount Q-value in Equation 4. The term $\max_{a'} Q(s_n, a')$ in Equation 4 is a scalar value and we cannot use discount vector $\mathbf{\Gamma}$ directly for the discounting. To preserve the temporally different objectives, we introduce a function to determine the discount factor based on the obtained reward vector. The function is named *reward dependent discount function* $f : \mathbb{R}^k \mapsto [0, 1]$. Using f, we can preserve the temporally different objectives because the discount value can be changed. Finally, we can write the multi-discount Q-learning equation as follows.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[\sum_{\tau=0}^{n-1} \sum_{i=0}^{k} \gamma_i^{\tau} r_{t+\tau,i} + f(\mathbf{r_{t+\tau}})^n \max_{a'} Q(s_{t+n}, a') - Q(s_t, a_t)] \quad (5)$$

Multi-discount Deep Q-learning (MD-DQN): Deriving the Multi-discount Deep Q-learning from Equation 5 is straightforward. We can define the *target function* in Equation 3 for MD-DQN with *n-step* return as shown in the below equation. Then, this can be used to compute the loss function in Equation 2 by replacing y_t^{DQN} with y_t^{mo}.

$$y_t^{mo} = \sum_{\tau=0}^{n-1} \sum_{i=0}^{k} \gamma_i^{\tau} r_{t+\tau,i} + f(\mathbf{r_{t+\tau}})^n \max_{a'} Q(s_{t+n}, a'; \theta_t^-)$$

A. Trajectory Optimization as an MDP

We now formulate the trajectory control and cruise control problem as an MDP. For each task there is a separate reward.

States: The state space consists of six states. First three states are related to trajectory control and the rest are related to cruise control. The first state is the current speed of the vehicle. The second is the distance to the intersection region. The third is the remaining time to reach the intersection region. The fourth is the front vehicles speed. The fifth is the distance between the vehicle and the front vehicle. The last one is the acceleration of the front vehicle.

Actions: The action space consists of three actions; 1) brake action, 2) no-op action, 3) acceleration action. The actual values assigned to these actions are -1, 0, 1, respectively. A similar set of actions were used for vehicle speed control [28].

Reward vector: The reward vector in this problem is a 2-d vector $\mathbf{r} = \{r_1, r_2\}$ where r_1 and r_2 are the rewards received for the cruise and trajectory control tasks respectively. Note that all the numerical values used below for rewards were selected experimentally to balance both tasks.

In the trajectory control task, the below reward is given at the end of an episode based on whether the vehicle reached the intersection starting point at the scheduled time.

$$r_{1,end} = \begin{cases} 10 + 3 * s_v(T_s), & \text{reached the intersection at } T_s \\ -10 & \text{otherwise} \end{cases}$$

where T_s is the scheduled time and $s_v(T_s)$ is the speed of the vehicle at the end of the episode. This reward encourages the vehicle to reach the intersection exactly at the scheduled time at high speed.

Besides $r_{1,end}$ at the end of an episode, at every time step t, $r_{1,step}$ is given as the negative value of the vehicle position normalized by the control region length.

$$r_{1,step} = -x_v(t)/L \quad \text{at every time step} \quad (6)$$

This encourages the vehicle to stay closer to the intersection and allows vehicles that follow enough space to enter the control region.

In the cruise control task, r_2 is assigned based on the gap between the front vehicle. r_2 encourages the vehicle to keep a safe gap whenever possible. A large negative reward is given to avoid crashing with the front vehicle.

$$r_2 = \begin{cases} -400 & \text{if a vehicle crashed with the front vehicle} \\ 0.1 & \text{else if } 6m < gap < 20m \\ -0.1 & \text{else if } gap < 6m \\ 0 & \text{otherwise} \end{cases}$$

The constant values used in rewards ($r_{1,step}, r_{1,end}$, and r_2) are obtained through changing the scale of these rewards and observing the vehicle behaviour. With the currently chosen reward constants, our method can achieve a good balance between trajectory control and cruise control. Note that MD-DQN claims described Section VI are valid independent of value of these constants.

Multi-discount Q-learning for trajectory control: We formulate MD-DQN for the AIM problem as follows. We denote discount vector $\mathbf{\Gamma}$ as $\{d_1, d_2\}$ and the reward dependent discount function, $f(\mathbf{r})$ defined as d_2 when r_2 is non-zero and d_1 when r_2 is zero [3].

VII. Experimental Methodology

We conducted three types of experiments following the experimental setups used in the literature [8], [16]. The first set of experiments evaluate the MD-DQN against Q-learning. The next set of experiments evaluate the trajectory computed by MD-DQN compared to an optimal policy computed using linear programming. Since MD-DQN computes the trajectory, these experiments are designed to compare the trajectory computed by MD-DQN with other baselines and to show the trajectory is near-optimal. The last set of experiments is designed to evaluate the overall impact of CMQ-AIM in an intersection compared to other baselines in terms of travel time and safety. We conduct the above experiments with a microscopic traffic simulator, SMARTS [29] using two experimental setups discussed next. The first setup is used to evaluate the MD-DQN Agent in the first and second experiments mentioned above, and the second setup is used for the third experiment mentioned above.

Vehicle Experimental Setup (VE-setup): This setup includes two vehicles (*leading vehicle* and *following vehicle*) that are arriving to an intersection on the same road. The *following vehicle* is given a time (chosen uniformly at random) to reach the intersection, and is controlled by the RL-based Agent. The *leading vehicle* can be either an autonomous or a human-driven vehicle (in which the RL-based Agent does not control the behavior).

Intersection Experimental Setup (IE-setup): This experimental setup includes a 4-legged intersection with 2-lane roads. Vehicle arrivals are modeled as a Poisson distribution. Whenever a new vehicle arrives, the polling system computes a time-schedule for every vehicle, and then, RL-based Agents per-vehicle are used to control the vehicle speed.

Baselines: First, to compare the effectiveness of multi-objective learning of the RL-based Agent, we use Threshold Lexicographic DQN (TLDQN) proposed by Li *et al.* [23]. We represent the trajectory and cruise control tasks as two separate Q-networks with lexicographic orders. At every time step, possible actions with higher Q-values are selected from each Q-network, and then the final action is chosen by considering the lexicographic order.

For the trajectory optimization task, we use two baselines. First, we use the same linear program formulation adopted by Miculescu *et. al* [8] for the trajectory control task to find the optimal solution. We denote this baseline as **LP-AIM**. Second, we use a heuristic solution for the trajectory optimization task, similar to the one proposed by Au *et al.* [20]. In Au *et al.* the objective is to optimize the time to reach the intersection. In contrast, in our trajectory optimization task, we maximize the end velocity and minimize the value in Equation 6. We denote this baseline as **H-AIM**.

For the schedule optimization task (the coordinating agent), we use traditional First-Come-First-Serve scheduling [4] as a baseline. We denote this baseline as **FCFS-AIM**.

A. Evaluation Metrics

We evaluate our solution based on the following metrics.

Trajectory Performance: The quality of trajectory trj, can be measured by computing how close the vehicle was to the intersection throughout the trajectory, leaving enough space for the vehicles behind. This can be computed as $X(trj) = \sum_{t=0}^{T_s} x_v(t)_{trj}/L$ where $x_v(t)_{trj}$ is the position of the vehicle at time t. The lower $X(trj)$ indicates that the vehicle has stayed closer to the intersection, which is better as described in Section V-B. Equation 7 can be used to compare two trajectories (trj_1 and trj_2).

$$diff(trj_1, trj_2) = |X(trj_1) - X(trj_2)| \tag{7}$$

Because the travel time depends only on the Polling-based Coordinating Agent, the travel time is an unsuitable metric to compare LP and MD-DQN in **VE-setup**. Thus, the above metric is used for the comparison.

Total Average Travel Time: The total travel cost defined in Section IV is used in **IE-setup** to calculate the travel time reduction for the entire simulation period.

B. Parameter Settings

The following parameters have been set: max. speed is $80kmh$, max. acceleration/deceleration is $2ms^{-2}$ and the discrete simulation step size Δt is $0.2s$ which are the default values of SMARTS. The control length L is set to 400m, similar to the previous work [8]. The minimum value of *service* and *switch over* times in the queue transition function are set to 1 second so that the vehicles have enough time to cross the conflicting squares (See Figure 1b) at the maximum speed. For vehicles with higher length ($l_{v,i}$), *service* time is however increased to allow a safe traversal.

While training the RL-based Agent (using [30]), the learning rate is set to 1e-5, exploration factor ϵ is annealed from 1 to 0 over 120000-time steps. The discount factor γ is in the range of 0.9-1. The RL parameters are selected through a parameter sweep. i.e. the hyperparameters in RL are selected by observing the reward graphs (the convergence rate and the maximum reward). The primal-dual gap of the Gurobi solver is set to 1-e5 to avoid high computation times that

[3] Here, d_1, d_2 are selected as 0.9 and 1 through a parameter sweep

occur when finding an optimal solution. The experiments are conducted in a server with an Intel Xeon(R) processor, 24 GB RAM, and an Nvidia GRID P40 GPU.

VIII. Experimental Results

Multi-discount Q-learning vs Q-learning: This experiment evaluates the advantage of multi-discount Q-learning over traditional Q-learning. The experiments are conducted with the **VE-setup** described in Section VII. The RL-based Agent is trained to control the *following vehicle* to reach the intersection exactly at the scheduled time while avoiding collision with the *leading vehicle*. During the training process, scheduled times are assigned uniformly at random at the start of an episode. To simulate short-term driving behavior, the *leading vehicle* selects uniformly at random whether to accelerate, decelerate or maintain the speed every 2 secs.

Figure 2 shows the reward achieved over 360000 time-steps (until the rewards converged) by each algorithm. Here, DQN-1 and DQN-0.9 refer to Q-learning with fixed discount factors of 1 and 0.9 respectively which were selected through a parameter sweep. Figure 2a shows the total episodic reward (combining rewards from the trajectory control and cruise control). There is a clear gain in the total reward of MD-DQN compared to fixed DQNs, and TLDQN which highlights the importance of MD-DQN i.e. the higher total reward means that MD-DQN achieved both trajectory control and cruise control objectives better than the other baselines.

Next, to analyze the agent behaviour further, we represent the reward in each sub-task using $r_{1,end}$ and r_2 in Figure 2b and Figure 2c respectively. Let us first compare fixed DQNs with MD-DQN. In Figure 2c, DQN-1 converges to 0 because with the discount factor 1, DQN-1 is unable to learn to closely follow the leading vehicle, i.e. the short-term behavior of the *leading vehicle*, and thus it keeps a large safe distance from the *leading vehicle*. Due to keeping a large distance, DQN-1 did not reach the intersection at the scheduled time which resulted in a lower gain for $r_{1,end}$ in trajectory control.

In Figure 2c, DQN-0.9 achieves positive values as it can predict the *leading vehicle's* behavior. However, in Figure 2b, DQN-0.9 does not achieve a higher reward as it only sees a few time-steps ahead. Thus, DQN-0.9 does not have enough time to reach the intersection at maximum speed or exactly at the scheduled time. Therefore, the reward is less than what is achieved by MD-DQN. MD-DQN overcomes these limitations and achieves higher rewards in both Figure 2c and Figure 2b.

TLDQN is not able to achieve higher rewards than MD-DQN as shown in Figure 2. The performance of TLDQN is similar to DQN-0.9 in all the sub-figures. This is due to TLDQN learns tasks independently and only combines the actions from each objective at the inference time, thus failing to learn the inter-relationship between objectives. In contrast, MD-DQN is able to learn such relationships resulting in higher rewards.

Traffic level (# vehicles)	530	1080	1750
Dynamic Traffic Signal (DTS)(s)	77.59	235.73	543.18
FCFS-AIM(s)	15.21	131.46	388.57
H-AIM(s)	15.25	98.09	313.34
LP-AIM(s)	13.85	94.92	304.44
CMQ-AIM(s)	13.66	92.76	302.63

TABLE I: Total average travel time for dynamic traffic signal, LP-AIM, H-AIM and CMQ-AIM for different traffic levels.

Traffic level (# vehicles)	530	1080	1750
H-AIM	65	49	69
LP-AIM	1	66	51
CMQ-AIM	0	0	0

TABLE II: The number of vehicles failed to arrive to the intersection crossing point within 1 second from the scheduled time.

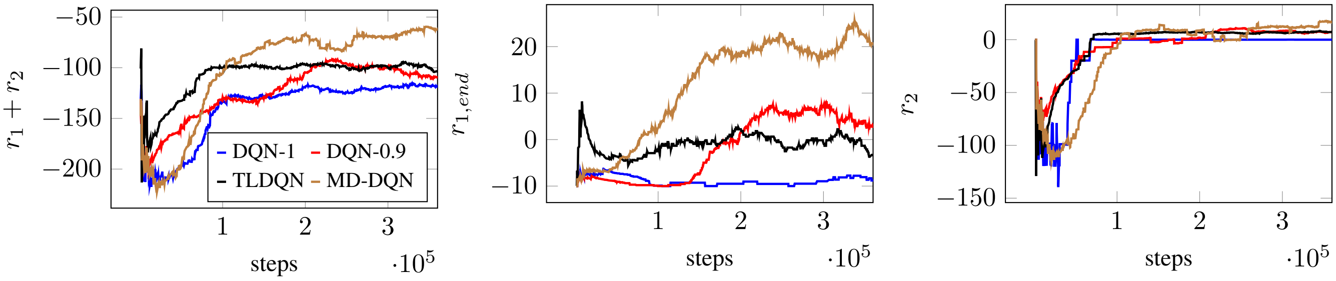
Multi-discount Q-learning vs Optimal Control: This experiment is focused on the optimality of multi-discount Q-learning. We use the same **VE-setup** used in the previous section and evaluate MD-DQN against linear programming (LP). As described in Section V-B, with LP the complete trajectory of the *leading vehicle* should be available for the *following vehicle* to compute its trajectory. Due to this reason, we can only conduct the experiments with the autonomous leading vehicle. For the comparison, we use the same LP formulation as in [8] and implemented using Gurobi. The pre-trained RL-based Agent from the previous section is used.

Figure 3a shows the trajectory performance of the trajectory computed by LP and MD-DQN. Both LP and MD-DQN are given the same scheduled times in the range of 20s to 32s, where 20s is the shortest possible time that a vehicle can traverse the control region length (400m) and 32s is the highest time scheduled by the intersection controller. The resulting trajectory performance value X is shown in the figure. Note that $X(\text{MD-DQN})$ achieves trajectory performance closer to the LP $X(LP)$ as the difference between trajectories, $diff(\text{MD-DQN}, LP)$ is less than 5 in majority of cases. This means that throughout the entire trajectory in control region of length L (which is 400m) the total vehicle position difference is 5m, which is a small value compared to L. Thus, MD-DQN is able to achieve near-optimal performance. As we show later, these minor deviations work in favor of the safety of the final solution.

Figure 3b shows the computational time required by LP and MD-DQN to select an action at different scheduled times. MD-DQN action selection is *two orders of magnitude faster* on average compared to that of LP. The maximum computation time of LP is 3.37 seconds. This is significant, since a vehicle traveling at 80 kmph may travel around 60m by the time the computation completes with LP, thus making the computed trajectory unsafe or unfeasible. In MD-DQN, a vehicle will only travel 0.08m during the action selection, making the RL approach safe.

This experiment highlights the benefit of MD-DQN in a real-time setting. With a small reduction in optimality, we are able to gain much lower computation time, which enables real-time AIM.

CMQ-AIM Evaluation: This experiment focuses on the overall performance gain of CMQ-AIM in reducing travel

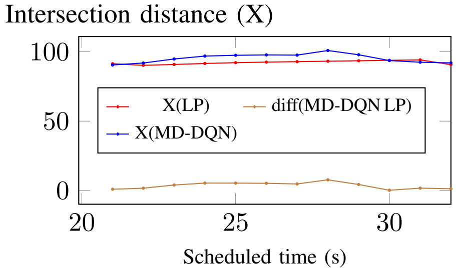


(a) Total episodic reward: $(r_1 + r_2)$

(b) Trajectory control reward

(c) Cruise control reward (r_2)

Fig. 2: The reward (moving average with window size of 10) achieved with fixed discount vs multi-discount Q-learning.



(a) Trajectory: LP vs MD-DQN

	MD- DQN	LP
average	8.3e-4	0.65
minimum	4.5e-4	0.03
maximum	4.2e-3	3.37

(b) Execution times (s) for MD-DQN and LP action selection. The average, minimum and maximum are shown.

Fig. 3: Performance vs execution times for LP and MD-DQN

time, using the **IE-setup**. As a baseline, SMARTS's Dynamic Traffic Signals (DTS) are used.

Table I presents the total average travel time for CMQ-AIM based solutions and dynamic traffic signal control with 30 minutes of simulation for different traffic levels (number of vehicles) using a Poisson distribution as in Hu *et al.* [31]. Regardless of the traffic level, AIM-based methods result in substantially lower average travel time compared to the DTS. This is due to avoiding vehicles' stop-and-go nature and yellow signal time. CMQ-AIM is substantially better than the traditional FCFS-AIM. This is because CMQ-AIM achieves a platooning-like behavior unlike the traditional FCFS-AIM where vehicles traverse the intersection in the order they arrived to the control region avoiding larger delays in switch over times while shifting from a road segment to a road segment. H-AIM, LP-AIM (the trajectory computed by LP), and CMQ-AIM result in similar travel times because these three methods use the same polling-based schedule controller proposed in Section V-A, which has a crucial impact on the total travel time. The slight differences in travel time between H-AIM, LP-AIM and CMQ-AIM are due to the deviations in trajectories discussed next.

Even though these minor deviations do not have a substantial impact on the travel time, they pose a significant safety risk. Table II shows the number of vehicles that did not arrive to the intersection crossing point within 1 second of their scheduled time. Since the switch over time is 1 second, any vehicle that has deviated more than 1 second poses a safety risk of crashing with a vehicle coming from another lane. Both heuristic-based and LP methods optimize the trajectory based on an internal model or a set of linear equations. However, these models and equations may not necessarily match the simulation dynamics. In the simulation environment, vehicles tend to have slight deviations from the intended path computed by LP due to changes in traffic. Due to these reasons, heuristic and LP-based methods tend to have higher deviation percentages, violating the switch over time and thus posing a safety risk. In contrast, RL-based CMQ-AIM achieves superior performance with no deviations, i.e. not a single vehicle violated the scheduled time throughout the entire simulation. This added safety level for RL-based Agents is attributed to the fact they are trained in the same simulation environment, allowing it to learn the vehicle and simulation dynamics (i.e. more intelligent). Further, since RL works iteratively, the RL-based Agent can adjust the vehicle acceleration even when vehicles are slightly deviating from the intended trajectory, thereby offering higher safety levels.

IX. Conclusion

This paper presents a real-time deployable AIM solution that uses a Polling-based Coordinating Agent and a set of distributed RL-based Agents. We propose a novel RL algorithm named *multi-discount Q-learning* to effectively achieve the AIM task intelligently in a complex intersection. We demonstrate that MD-DQN can successfully learn tasks with different temporal objectives, while the existing state-of-the-art approaches fail to do so. Our experimental results show that our proposed solution is safe, computationally efficient and close to optimal, making real-time autonomous intersection management deployable in real-world road networks.

References

[1] F. Carton, D. Filliat, J. Rabarisoa, and Q. C. Pham, "Evaluating robustness over high level driving instruction for autonomous driving," in *2021 IEEE Intelligent Vehicles Symposium (IV)*, 2021, pp. 129–135.

[2] F. Ye, P. Wang, C.-Y. Chan, and J. Zhang, "Meta reinforcement learning-based lane change strategy for autonomous vehicles," in *2021 IEEE Intelligent Vehicles Symposium (IV)*, 2021, pp. 223–230.

[3] A. P. Capasso, P. Maramotti, A. Dell'Eva, and A. Broggi, "End-to-end intersection handling using multi-agent deep reinforcement learning," in *2021 IEEE Intelligent Vehicles Symposium (IV)*, 2021, pp. 443–450.

[4] K. Dresner and P. Stone, "A multiagent approach to autonomous intersection management," *Journal of Artificial Intelligence Research*, vol. 31, pp. 591–656, 2008.

[5] D. Fajardo, T.-C. Au, S. T. Waller, P. Stone, and D. Yang, "Automated intersection control: Performance of future innovation versus current traffic signal control," *Transportation Research Record*, vol. 2259, no. 1, pp. 223–232, 2011.

[6] D. Carlino, S. D. Boyles, and P. Stone, "Auction-based autonomous intersection management," in *IEEE Conference on ITS*, 2013, pp. 529–534.

[7] M. R. Hafner, D. Cunningham, L. Caminiti, and D. Del Vecchio, "Cooperative collision avoidance at intersections: Algorithms and experiments," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1162–1175, 2013.

[8] D. Miculescu and S. Karaman, "Polling-systems-based autonomous vehicle coordination in traffic intersections with no traffic signals," *IEEE Transactions on Automatic Control*, vol. 65, no. 2, pp. 680–694, 2020.

[9] A. Colombo and D. Del Vecchio, "Efficient algorithms for collision avoidance at intersections," in *ACM International Conference on Hybrid Systems: Computation and Control*, 2012, p. 145–154.

[10] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley, "A survey of multi-objective sequential decision-making," *Journal of Artificial Intelligence Research*, vol. 48, pp. 67–113,, 2013.

[11] W. B. Knox and P. Stone, "Learning non-myopically from human-generated reward," in *International Conference on Intelligent User Interfaces*, ser. IUI '13, 2013, p. 191–202.

[12] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *IEEE International Conference on Robotics and Automation*, 2017, pp. 3389–3396.

[13] Y. Zhang, A. A. Malikopoulos, and C. G. Cassandras, "Decentralized optimal control for connected automated vehicles at intersections including left and right turns," in *IEEE 56th Annual Conference on Decision and Control*, 2017, pp. 4428–4433.

[14] A. Colombo and D. Del Vecchio, "Least restrictive supervisors for intersection collision avoidance: A scheduling approach," *IEEE Transactions on Automatic Control*, vol. 60, no. 6, pp. 1515–1527, 2015.

[15] G. R. de Campos, F. Della Rossa, and A. Colombo, "Optimal and least restrictive supervisory control: Safety verification methods for human-driven vehicles at traffic intersections," in *IEEE Conference on Decision and Control*, 2015, pp. 1707–1712.

[16] Y. Xu, H. Zhou, T. Ma, J. Zhao, B. Qian, and X. Shen, "Leveraging multiagent learning for automated vehicles scheduling at nonsignalized intersections," *IEEE Internet of Things Journal*, vol. 8, no. 14, pp. 11 427–11 439, 2021.

[17] Y. Wu, H. Chen, and F. Zhu, "Dcl-aim: Decentralized coordination learning of autonomous intersection management for connected and automated vehicles," *Transportation Research Part C: Emerging Technologies*, vol. 103, pp. 246–260, 2019.

[18] F. Altché and A. de La Fortelle, "Analysis of optimal solutions to robot coordination problems to improve autonomous intersection management policies," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, 2016, pp. 86–91.

[19] Q. Lu and K.-D. Kim, "Intelligent intersection management of autonomous traffic using discrete-time occupancies trajectory," *Journal of Traffic and Logistics Engineering Vol*, vol. 4, no. 1, pp. 1–6, 2016.

[20] T. Au, M. Quinlan, and P. Stone, "Setpoint scheduling for autonomous vehicle controllers," in *IEEE International Conference on Robotics and Automation*, 2012, pp. 2055–2060.

[21] A. Edwards, M. Littman, and C. Isbell, "Expressing tasks robustly via multiple discount factors [online]," 2015. [Online]. Available: https://www.semanticscholar.org/paper/Expressing-Tasks-Robustly-via-Multiple-Discount-Edwards-Littman/3b4f5a83ca49d09ce3bf355be8b7e1e956dc27fe

[22] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[23] C. Li and K. Czarnecki, "Urban driving with multi-objective deep reinforcement learning," in *International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '19, 2019, p. 359–367.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[25] H. Takagi, "Queuing analysis of polling models," *ACM Comput. Surv.*, vol. 20, no. 1, p. 5–28, Mar. 1988.

[26] J. Gregoire and E. Frazzoli, "Hybrid centralized/distributed autonomous intersection control: Using a job scheduler as a planner and inheriting its efficiency guarantees," in *IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 2549–2554.

[27] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 2nd ed. MIT Press, 1998, vol. 135.

[28] C. Desjardins and B. Chaib-draa, "Cooperative adaptive cruise control: A reinforcement learning approach," *IEEE Transactions on ITS*, vol. 12, no. 4, pp. 1248–1260, 2011.

[29] K. Ramamohanarao, H. Xie, L. Kulik, S. Karunasekera, E. Tanin, R. Zhang, and E. B. Khunayn, "SMARTS: Scalable microscopic adaptive road traffic simulator," *ACM Trans. Intell. Syst. Technol.*, vol. 8, no. 2, 2016.

[30] M. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli, S. Henderson, A. Novikov, S. G. Colmenarejo, S. Cabi, C. Gulcehre, T. L. Paine, A. Cowie, Z. Wang, B. Piot, and N. de Freitas, "Acme: A research framework for distributed reinforcement learning," *arXiv preprint arXiv:2006.00979*, 2020.

[31] M.-B. Hu, R. Jiang, R. Wang, and Q.-S. Wu, "Urban traffic simulated from the dual representation: Flow, crisis and congestion," *Physics Letters A*, vol. 373, no. 23-24, pp. 2007–2011, 2009.