2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Decentralized Conflict Resolution for Multi-Agent Reinforcement Learning Through Shared Scheduling Protocols

Tyler Ingebrand*, Sophia Smith*, and Ufuk Topcu

Abstract— **Decentralized multi-agent reinforcement learning (MARL) is an inherently difficult problem because agents can have individual, unique objectives and no direct incentive to cooperate. Conflicts often arise over bottlenecks in the environment, such as a shared key or an intersection, where multiple agents need to access a single resource. To resolve these conflicts, we propose the use of a shared scheduling protocol. A scheduling protocol coordinates agent behavior such that one agent is allowed to greedily use the resource while the others are required to wait. In particular, we are interested in decentralized scheduling protocols that can be implemented independently by each agent without a centralized controller. We present three protocols and prove that they resolve conflicts when obeyed by all agents. In training, agents learn to obey the protocol as violations incur a penalty. Experimental results show that scheduling protocols increase the performance of multi-agent training fivefold compared to baseline decentralized MARL.**

I. Introduction

Without intervention, agents in decentralized multi-agent reinforcement learning (MARL) lack the information and incentives to cooperate. Many real-world environments are crowded with many agents, such as warehouse robots, self-driving cars, and autonomous delivery drones. These applications require agents to coordinate access to shared resources that are necessary to achieve high reward. However, decentralized MARL agents learn individual policies that greedily accomplish their own tasks. Additionally, optimal policies depend on environment dynamics, which depend on other agents' behavior. Without privileged knowledge of other agents' policies or communication, coordination is impossible.

Centralized MARL simplifies coordination over resources but fails for several reasons. Centralized control considers a multi-agent system to be a single agent, making coordination and cooperation trivial. However, centralized control scales poorly with the number of agents due to the curse of dimensionality [1]. Additionally, centralized control might be impossible when some of the agents cannot be centrally controlled or when agents refuse to allow a centralized controller to dictate their actions. For these reasons, learning coordination through decentralized approaches is necessary.

This work was supported by the Army Research Office (ARO W911NF-20-1-0140) and the Air Force Office of Scientific Research (AFOSR FA9550-22-1-0403).

T. Ingebrand is with the Chandra Department of Electrical and Computer Engineering, S. Smith is with the Oden Institute for Computational Engineering and Sciences, while U. Topcu is with the Oden Institute and the Department of Aerospace Engineering and Engineering Mechanics at the University of Texas at Austin, Austin, TX, USA. Email: {tyleringebrand, smith.sj, utopcu}@utexas.edu.
* Equal contribution.

We propose the use of shared scheduling protocols to resolve conflicts over resources in decentralized MARL. We provide a formal definition for bottlenecks, which are a shared resource that many agents need to use but only a subset can use at a time. Scheduling protocols allocate the bottleneck resource to a subset of agents while requiring all other agents to wait. We train agents to follow a protocol by appending an additional signal to their state space and penalizing agents for breaking the rules. We prove that scheduling protocols resolve conflicts, allowing all agents to use the resource during an episode. As a result, all agents are able to achieve high rewards simultaneously.

In particular, we are interested in decentralized scheduling protocols as they can be implemented independently by each agent. Decentralized scheduling protocols allocate resource access to agents without explicit agent coordination. This means that no centralized controller is necessary to resolve conflicts. Decentralized protocols only require the history of an individual agent to determine if that agent should access a bottleneck or wait, as shown in Fig. 1.

We apply three pre-existing protocols from the networking and cooperating system communities to the decentralized MARL setting and prove they resolved conflicts. Carrier sense multiple access with collision detection (CSMA/CD) [2] is a decentralized media access control protocol for sharing an Ethernet wire, while round robin (RR) and shortest remaining time first (SRTF)[3] are job scheduling protocols for processors. We prove that RR, CSMA/CD, and SRTF satisfy the definition of a resource scheduling protocol and therefore are able to resolve conflicts in a MARL setting. CSMA/CD and SRTF are both decentralized protocols while RR is centralized.

We find scheduling protocols results in a five-fold increase in the number of agents converging to successful policies during decentralized training. To evaluate the scheduling protocols, we run experiments on crowded gridworld environments with 15 agents learning simultaneously. Experimental results show that decentralized independent learners fail to converge as agents compete over the bottleneck resources and mutually prevent task completion. In contrast, agents learning with a scheduling protocol simultaneously converge to task completion.

II. Related Works

Many MARL algorithms that can learn cooperative behavior utilize inter-agent communication [4, 5, 6, 7]. In some circumstances, it is possible to provide convergence guarantees given that each agent can access the full state

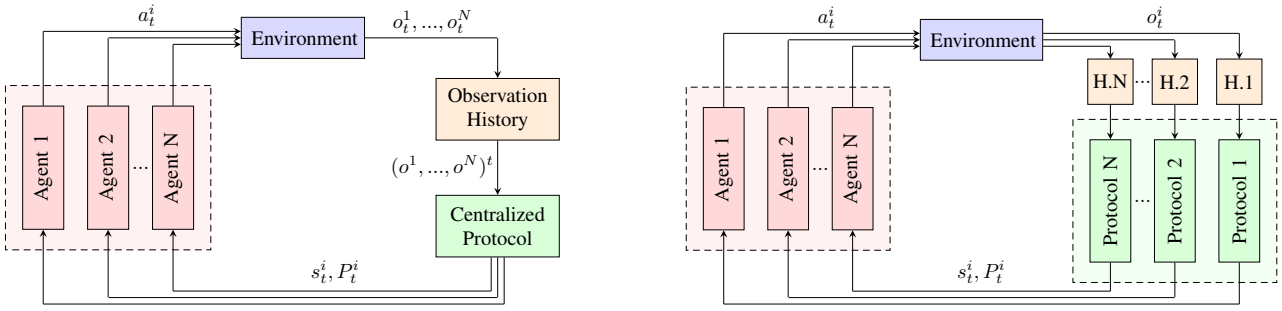

Fig. 1: Comparison of information flow in centralized (a) and decentralized (b) scheduling protocols for decentralized MARL. A centralized protocol takes in the observation history for all agents and generates the local state and protocol output for each agent. A decentralized protocol only requires the history of agent i to generate the local state and protocol output for agent i. Note that H.1 denotes the observation history for agent 1, H.2 denotes the observation history for agent 2, etc.

[6, 8]. However, requiring the full state to be globally visible might be intractable in multi-agent systems with many agents. In addition, communication algorithms assume all agents are optimizing the average network reward, rather than optimizing for their respective, individual rewards. [6, 8] In this work, we seek to achieve high total reward while only optimizing for individual rewards with local state information.

Another approach to learning cooperative behavior is to condense relevant global information into a smaller context variable. If the context variable provides enough information to make the transition function stationary from each agent's perspective, then it is possible for each agent to learn policies independently [9]. While independent learning is ideal, defining a global context variable which makes transitions stationary may be infeasible. In contrast, we use scheduling protocols to provide context variables that provably include enough information to share a resource without requiring stationary dynamics.

Shielding is a method of ensuring cooperation which formally verifies that selected actions satisfy some specification. Shields can be either centralized or factorized and require some knowledge of transition dynamics [10]. Ensuring cooperation over a long horizon for many agents may be computationally expensive and depends on knowledge about system dynamics. Scheduling protocols make no assumptions about the availability of transition dynamics.

III. Preliminaries

Throughout this paper, $\mathbb{R}$ refers to the reals, $\mathbb{N}$ refers to the naturals, $\mathbb{P}(\cdot)$ is the probability operator, and $\Delta(Y)$ denotes the set of all probability distributions over a set Y.

A. Reinforcement Learning (RL)

In the typical reinforcement learning setup, an agent interacts with an environment over an infinite time horizon. The environment is modeled as a Markov decision process (MDP) defined as a tuple $\langle S, A, T, R, \gamma \rangle$, where S is a state space, A is an action space, $T : S \times A \mapsto \Delta(S)$ is the transition function, $R : S \times A \times S \mapsto \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is a discount factor. Through interactions with the environment, an agent learns a policy $\pi : S \mapsto \Delta(A)$ and attempts to find an optimal policy π^* that maximizes the expected discounted future reward.

There are many prior works which describe how to find the optimal policy [11, 12, 13, 14, 15]. Each algorithm has different convergence guarantees, scalability, and potential to learn from offline data. Thus, the choice of algorithm depends on the setting. We use RL algorithms as subroutines which could be interchanged depending on the setting.

B. Multi-Agent Reinforcement Learning (MARL)

We model a multi-agent environment as a decentralized MDP with a finite number of agents [16].

Definition 1: A decentralized MDP (dec-MDP) for N agents is the tuple $M = \langle S, A, T, R, \Omega, O, \Gamma \rangle$, where

- S is a finite set of states,
- $A = A^1 \times A^2 \times \ldots \times A^N$ is a finite set of joint actions such that A^i is the set of local actions for agent i,
- $T : S \times A \mapsto \Delta(S)$ is the joint transition function,
- $R : S \times A \times S \to \mathbb{R}$ is a reward function such that $R(s, a, s')$ is the joint reward agents receive for taking action a in joint state s and transitioning to s',
- $\Omega = \Omega^1 \times ... \times \Omega^N$ is a set of joint observations with Ω^i being the set of local observations for agent i,
- $O : S \times A \times S \mapsto \Delta(\Omega)$ is the observation function, such that $O(s, a, s')$ is a probability distribution over observations and describes the probability of each agent i seeing observation $o^i \in \Omega^i$ after the taking joint action a in state s and transitioning to state s', and
- $\Gamma \in [0, 1)^N$ is a vector such that Γ^i, the ith entry of Γ, is the discount factor for agent i.

Dec-MDPs are *jointly fully observable*, meaning that for all joint observations $o \in \Omega$, there exists a joint state s such that $\mathbb{P}(s|o) = 1$.

A dec-MDP M is *factored* if the joint state space $S = S^1 \times S^2 \times ... \times S^N$, where each S^i is agent i's local state space. A factored dec-MDP is *locally fully observable* if for

all local observations o^i, there exists a state $s^i \in S^i$ such that $\mathbb{P}(s^i|o^i) = 1$. If a factored dec-MDP is locally fully observable, each agent has complete information about their local state.

A factored dec-MDP is *reward-independent* if there exists $R^1, ..., R^N$, $R^i : S^i \times A^i \times S^i \mapsto \mathbb{R}$ such that

$$R(s, a, s') = \sum_{i=0}^{N} R^i(s^i, a^i, s^{i\prime}).$$

For an agent i with local state, action s_t^i, a_t^i, and next local state s_{t+1}^i, the local reward $r^i = R_t^i(s^i, a_t^i, s_{t+1}^i)$.

For a factored dec-MDP M, a trajectory of states $\xi = s_0 s_1 ... s_N$ is a finite string of states $s_t \in S$ such that for all t, there exists joint action a_t such that $\mathbb{P}(s_{t+1}|T(s_t, a_t)) > 0$. For an individual agent i, a sequence of local states $\xi^i = s_0^i s_1^i ... s_n^i$ is called a local trajectory if there exists trajectory $\xi = s_0 s_1 ... s_N$ such that for each joint state s_t, s_t^i is the local state for agent i.

A local policy $\pi^i : S^i \mapsto \Delta(A^i)$ defines the probability of taking local action a^i in local state s^i. Local policies are fully decentralized and only use local state information to produce actions. An optimal local policy π^{i*} is a policy that optimizes the total expected discounted local reward $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t^i]$, where $\gamma = \Gamma^i$.

We consider environments where an agent may be *isolated* in certain states, meaning that the actions of other agents do not affect the transition for the isolated agent. We denote the action space for all agents except agent i as A^{-i}. With a slight abuse of notation, a transition function can be written as $T(s, a^i, a^{-i})$. Formally, at joint state s, an agent i is isolated if for all $a^i \in A^i$ and $a, a' \in A^{-i}$, $T(s, a^i, a) = T(s, a^i, a')$. An isolated optimal local policy $\pi^{\mathbf{i}*}$ is a local policy that maximizes the total expected discounted local reward given that agent i is isolated at every timestep. Isolation implies that the transition dynamics for an agent only depend on that agent's actions. This returns the environment to the single-agent setting. Therefore, isolated optimal local policies are decentralized and may be found via an RL subroutine.

We assume that isolated optimal policies act as a desired target for what an agent trained in a concurrent multi-agent setting should achieve. However, even with this assumption, isolated optimal policies tend to fail when executed in a concurrent multi-agent setting. Isolated policies are implicitly conditioned on the lack of inter-agent interactions, an assumption that is violated at execution time.

Instead, we focus on concurrent training, where agents interact with each other while learning their local policies. In this setting, the transition dynamics are dependent on other agents' policies making the environment non-stationary. Therefore, learning an optimal local policy π^{i*} is a moving target and does not typically have convergence guarantees [17]. Still, concurrent training is desirable because of its scalability and robustness to inter-agent interactions. The methods described below aim at improving the performance of concurrent, decentralized MARL in an environment with a bottleneck.

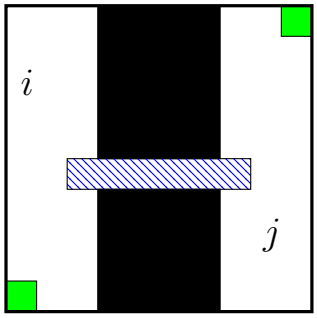

Fig. 2: The hallway environment, where agents must move from one room to the other. Two rooms are connected by a hallway with room. Green squares represent the target locations. Blue highlighted region makes up the bottleneck.

IV. Methods

In this section, we formally define a bottleneck, a shared resource over which conflicts are likely to occur. We use this definition to motivate the required properties for a scheduling protocol. Next, we present three protocols and prove they resolve conflicts between agents. Finally, we describe how scheduling protocols can be used to improve concurrent, decentralized MARL.

We first introduce an illustrative example, shown in Fig. 2, which we call the hallway environment [7]. This environment is made up of two rooms connected by a hallway. Agents randomly spawn in a room and are tasked with navigating through the hallway to the other room. The hallway is only wide enough for one agent to pass through at a time.

A. Critical Resources and Bottleneck Identification

We now define a critical resource for a factorized, locally fully observable, and reward-independent dec-MDP M.

Definition 2 (Critical Resource): For an agent i at state $s^i \in S^i$, critical resources $C^i(s^i) \subset S^i$ is the set of states such that $s \in C^i(s^i)$ if under any isolated optimal policy, a rolled-out local trajectory from state s^i that achieves optimal reward will include state s.

Intuitively, the critical resources of agent i are states that are mandatory for an agent to receive optimal reward. Critical resources can be found by training each agent in the environment alone to learn multiple isolated optimal policies.

In the hallway environment, the critical resources for agent i include the target location in the opposite room and the hallway states they must pass through, highlighted in Fig. 2. Observe that an agent's critical resources depend on its current state. For example, once an agent enters the opposite room in the hallway environment, rolled out trajectories that reach the target no longer include the hallway.

Given a definition for critical resources, we define what it means that an agent can block another agent from using a critical resource. Agent i at state s^i is *blocking* agent j at s^j if for as long as agent i is at state s^i, there exists a critical

resource $c \in C^j(s^j)$ such that there does not exist a local trajectory ξ^j starting in s^j and ending in c. An agent i at state s^i *can block* agent j at s^j *via* state $s' \in S^i$ if there exists a local trajectory ξ^i from s^i to s' such that agent i at state s' is blocking agent j.

Next, we define a dec-MDP bottleneck, which consists of states where conflicts between agents are likely to arise.

Definition 3 (Bottleneck): For dec-MDP M with each agent k at local state s^k and critical resources $C^k(s^k)$, a bottleneck is the set of states $B(s^1, \ldots, s^N)$, where state $b \in B(s^1, \ldots, s^N)$ if and only if:

1) there exists agents i, j, $i \neq j$, such that $b \in C^i(s^i) \cap C^j(s^j)$, and
2) there exists agents $i, j, j \neq i$ such that agent j can block agent i at state s^j via b.

A bottleneck is the set of critical resources which at least two agents have in common and at least one agent can be prevented from using due to the state of another agent. In the hallway environment, agents i and j can block each other from accessing the opposite room by standing in the hallway. Therefore, the hallway states form a bottleneck, as highlighted in Fig. 2. Observe that a bottleneck depends on the states of the agents. If less than two agents need a state, or no agent can be prevented from reaching that state by another agent, then that state is no longer part of a bottleneck.

Bottleneck states can be grouped into individual bottleneck sets consisting of bottleneck states that are fully connected via dec-MDP transitions. Bottleneck states $B(s^i, ..., s^j)$ are fully connected if for all $b, b' \in B(s^i, ..., s^j)$, there exists for each agent k a local trajectory $\xi^k = bs_1^k s_2^k \ldots s_n^k b'$ with $s_1^k, ...s_n^k \in B(s^i, ..., s^j)$. Fully connected bottlenecks can be considered a single resource and will be managed by a single instantiation of a protocol. If there are multiple fully connected bottlenecks, each one will be managed separately. For the remainder of this work, we assume there is a single fully connected bottleneck in the environment.

As an episode progresses, an agent may no longer need any resources in the bottleneck. An agent k at local state s_t^k *passes through* a fully connected bottleneck $B(s_t^1, ..., s_t^N)$ at time t if $s_t^k \notin B(s_t^1, ..., s_t^N)$ and for all $b \in B(s_0^1, ..., s_0^N)$, $b \notin C^k(s_t^k)$. Agents have passed through the bottleneck if the bottleneck no longer includes states that are necessary for the agent to achieve an optimal reward.

We assume that for any fully connected bottleneck, there exists a finite *necessary time* that describes the minimum number of timesteps needed for an agent to pass through the bottleneck. Specifically, if $\beta^i \in \mathbb{N}$ is the necessary time for agent i at state s_t^i, then there exists a local trajectory $\xi^i = s_t^i s_{t+1}^i ...$ with length less than $\beta^i + 1$ such that agent i passes through the bottleneck at time $t + \beta$. We assume that as long as agent i is not being blocked by another agent, the necessary time β^i only depends on state s^i, not on other agents. We refer to the maximum β^i over all agents and over all states as β.

For a fully connected bottleneck $B(s_t^1, ..., s_t^N)$, we say agent i has a *conflict* at time t if agent i has not passed through the bottleneck. A conflict implies more than one agent needs to use the bottleneck resource but they cannot use it at the same time. Concurrent decentralized MARL will often fail to resolve conflicts. In the next section, we formally define scheduling protocols to coordinate agent behavior.

Although it is possible to identify bottlenecks by following the definitions above, it is also likely that bottlenecks may be identified by understanding the environment and predicting where agents are likely to interact. Alternatively, multiple agents can be concurrently trained in the environment which may demonstrate conflicts.

B. Scheduling Protocol Requirements

A scheduling protocol is an algorithm that coordinates access to a bottleneck.

Definition 4 (Scheduling Protocol): For bottleneck $B(s^1, ..., s^N)$ and necessary time β, protocol $P = P^1 \times ... \times P^N$ at time t with $P^i : (\Omega^i)^t \mapsto \{go, wait\}$. Here $(\Omega^i)^t$ are sequences of local dec-MDP observations of length t. For all agents i, there exists a interval of time $[t^i, t^i + \beta]$ such that for all $t' \in [t^i, t^i + \beta]$,

$$P^j(o_0^j ... o_{t'}^j) = \begin{cases} go & \text{if } j = i \\ wait & \text{if } j \neq i \text{ and } j \in X^i(s_{t'}^i), \end{cases}$$

where $X^i(s^i)$ is the set of agents that can block agent i. Interval of time $[t^i, t^i + \beta]$ is called agent i's *turn* under scheduling protocol P. If $j \neq i$ and $j \notin X^i(s_{t'}^i)$, then it is up to the implementation if $P^j(o_0^j ... o_{t'}^j) = go$ or $wait$.

Scheduling protocols coordinate access to a common state using an agent's local observations. In a locally fully observable dec-MDP, o^i fully determines the local state of agent i. However, o^i can contain additional information, such as whether or not a bottleneck is occupied.

When $P^i(o_0 ... o_{t'}) = go$, agent i should greedily access the bottleneck, however when $P^i(o_0 ... o_{t'}) = wait$, agent i should not access the bottleneck. Thus, protocols give each agent access to a bottleneck for the necessary time while other agents stay away.

A protocol is only required to tell agents to wait that are capable of blocking the agent whose turn it is. If an agent j can no longer block agent i during agent i's turn, a protocol can tell both agents to act. This occurs when two agents can use a resource simultaneously as long as one agent has a head start, such as when two agents are moving through the hallway in the same direction. In the event $X^i(s^i)$ is unknown, P satisfies the definition of a protocol if during agent i's turn, $P^j(o_0 ... o_{t'}) = wait$ for all agents $j \neq i$.

Proposition 1: Assume that for any agent i, i passes through the bottleneck at time $t + \beta$ if for all $t' \in [t, t + \beta]$, $P^i(o_0 ... o_{t'}) = go$ and $P^k(o_0 ... o_{t'}) = wait$ for all $k \in X^i(s_{t'}^i)$. Then, protocol P resolves conflicts for every agent.

Proof: By the definition of a protocol P, there exists an interval $[t^i, t^i + \beta]$ such that for all $t' \in [t^i, t^i + \beta]$, $P^i(o_0 ... o_{t'}) = go$ and $P^k(o_0 ... o_{t'}) = wait$ for any k such that k can block i. Then, by assumption agent i passes through the bottleneck. Therefore, scheduling protocol P resolves conflicts for agent i. Since P ensures each agents receives a turn, P resolves conflicts for all agents. ■

In their most general form, scheduling protocols are centralized. Centralized protocols require either the environment to coordinate protocol states, or a data processing step before agents make decisions. Protocols can also be decentralized, where each agent can implement the protocol independently without coordination. We present examples of both centralized and decentralized protocols in the next section. See Fig. 1 for a visualization.

C. Protocol Verification

We present three different protocols pulled from the networking and operating system communities and prove that they satisfy the definition of a scheduling protocol. For each protocol, $P = P^1 \times ... \times P^N$. At time t, $P^i : (\Omega^i)^t \mapsto \{go, wait\}$ for all agents i with local observation space Ω^i of a locally fully observable dec-MDP. 0

a) Round Robin (RR): RR is a turn-based protocol where a subset of agents are instructed to act greedily for a period of time. All other agents are told to wait. The subset of agents must be able to use the resource simultaneously without preventing each other from using the resource [3].

Formally, for dec-MDP M with bottleneck $B(s^1, ..., s^N)$, there exists an ordering $1, 2, ..., N$ such that if k is in the k^{th} position, then for all

$$t' \in [(k-1)(\beta+1), (k-1)(\beta+1)+\beta],$$

$$P^l(o_0^l ... o_{t'}^l) = \begin{cases} go & \text{if } l = k \\ go & \text{if } l \neq k \text{ and } l \notin X^k(s_{t'}^k) \\ wait & \text{if } l \neq k \text{ and } l \in X^k(s_{t'}^k). \end{cases}$$

Proposition 2: Round robin is a scheduling protocol for bottleneck $B(s^1, ..., s^N)$ with necessary time β.

Proof: Follows directly from the definition of a scheduling protocol, since each agent has a unique place n in the agent ordering and a turn during the time interval $[(n-1)(\beta+1), (n-1)(\beta+1)+\beta]$ of length β. ■

In round robin, local observations o_t^k must contain at a minimum the local state s_t^k and indication if it is agent k's turn in the ordering. Knowing that agent k cannot block agent i during agent i's turn may increase the time agent k has access to a bottleneck, but when this information is unavailable, $P^k(o_0^k ... o_{t'}^k) = wait$. Though an agent k can pass through the bottleneck during the turn of agent i if agent k cannot block agent i, agent k is still explicitly given their own turn.

While RR is easy to implement and understand, it requires a centralized authority to keep track of the ordering. Additionally, RR can be inefficient during training because each agent has relatively few timesteps to use the bottleneck resource each training episode, hindering exploration. However, equal access to the bottleneck at predictable time intervals may be favorable trade off.

b) Carrier Sense Multiple Access with Collision Detection (CSMA/CD): CSMA/CD is a first-come, first-serve protocol. When an agent begins using the critical resource, CSMA/CD dictates all other agents should wait. A *collision* occurs when two agents begin using the critical resource simultaneously. When a collision occurs, both agents should stop using the critical resource for a random amount of time. This randomness prevents collisions from repeatedly occurring, as one of the agents will eventually be allowed to access the bottleneck while the other must keep waiting [2].

Formally, for each agent i, there exists a counter function $f_t^i : (\Omega^i)^t \mapsto \mathbb{N}$ that counts the number of collisions agent i was involved in throughout the observation history $o_0^i ... o_t^i \in (\Omega^i)^t$. After agent i participates in a collision, it is required to wait for a random number of timesteps between 1 and $2^{f_t^i(o_0^i ... o_t^i)}$. This is known as exponential backoff. Let W_t^i be the amount of time an agent still needs to wait at time t before it is allowed to use the resource. CSMA/CD is then defined as:

$$P^i(o_0 ... o_t) = \begin{cases} wait & \text{if } W_t^i > 0 \\ wait & \text{if } W_t^i = 0, s_t^i \notin B(s^1, ..., s^N), \\ & \quad \text{and } \exists j : s_t^j \in B(s^1, ..., s^N) \\ go & \text{else.} \end{cases}$$

Proposition 3: Assume that for any agent i in the bottleneck, agent i at time t does not leave the bottleneck unless $P^i(o_0^i .. o_t^i) = wait$. Assume also that agent i passes through the bottleneck after β consecutive timesteps in the bottleneck. Then CSMA/CD is a scheduling protocol for bottleneck $B(s^1, ..., s^N)$ with necessary time β.

Proof: Observe that since agents comply to CSMA/CD, an agent outside the bottleneck will not enter if another agent is already there. Therefore, we consider the following two cases. The first case is if at time t, agent i enters the bottleneck and no other agents are in the bottleneck at timestep t. Then, $P^j(o_0 ... o_t) = wait$ for all other agents and $P^i(o_0^i ... o_t^i) = go$. Then, since agent i stays in the bottleneck for β timesteps, for all $t' \in [t, t+\beta]$, $P^j(o_0^j ... o_{t'}^j) = wait$ for all agents $j \neq i$ and $P^i(o_0 ... o_{t'}) = go$.

For the second case, suppose instead that two or more agents enter the bottleneck at time t. Then for each agent i in the bottleneck, W_t^i will be assigned a value between 1 and $2^{f_t^i(o_0 ... o_t)}$. Thus, these agents will receive $wait$ for some random number of timesteps. If for two agents i and j, $W^i = W^j$, they will both enter the bottleneck after their timers reach 0. However, as the number of collisions increase, the probability of two agents receiving the same random backoff time approaches 0. Thus, at least one agent will receive go while the others receive $wait$. This agent will enter the bottleneck alone, leading to the previous case. ■

At a minimum, the local observation at time t include an agent's local state, as well a signal indicating that another agent is in the bottleneck. Under the small assumption that each agent can detect when the bottleneck is in use, CSMA/CD is a fully decentralized protocol as each agent does not need any information about the protocol state of other agents. The local observation can include additional information to signal an agent if it is able to block the agent currently in the bottleneck. If an agent cannot block the agent currently in the bottleneck, it may use the bottleneck at the same time.

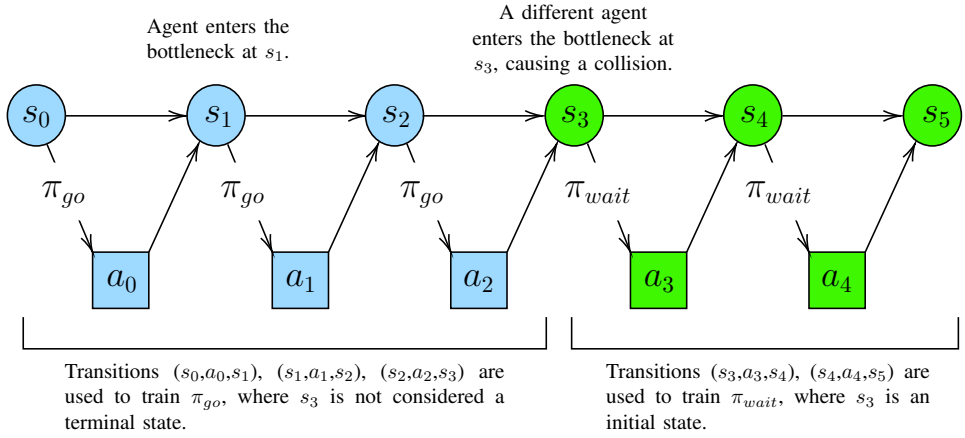

Fig. 3: An example trajectory from the perspective of an agent following the CSMA/CD protocol. Actions are a function of the state, shown as as an arrow with a corresponding policy label, and the next state is a function of the previous state and action, shown by two arrows. Blue states correspond to states where the protocol signals go and green states correspond to states where the protocol signals $wait$. The agent uses either π_{go} or π_{wait} to choose an action based on the current protocol state. For an action generated by either π_{go} or π_{wait}, the transition (s_i,a_i,s_{i+1}) is only used to train the respective policy. A change in protocol state is not considered a terminal transition.

In contrast to RR, CSMA/CD does not hinder learning as any agent may use the bottleneck when it is unused, providing ample opportunity to explore the bottleneck resource. However, even after training, collisions will occur forcing the agents involved to wait. In some environments, repeated backoffs may create exceedingly long waiting periods.

c) Shortest Remaining Time First (SRTF): SRTF is similar to CSMA/CD but prioritizes maximizing resource usage. If an agent enters the bottleneck, all other agents must wait. Should two agents enter the bottleneck simultaneously, the agent with the shortest remaining time necessary to pass through the bottleneck is instructed to act greedily while the other must exit the bottleneck. This procedure increases resource usage efficiency at the cost of reducing fairness. An agent that requires longer to use the bottleneck than other agents may repeatedly get turned away [3].

Formally, for an agent i at state $s_t^i \in B(s^1, ..., s^N)$, let $\beta^i(t)$ be the time remaining for agent i to pass through the bottleneck.

$$P^i(o_0^i ... o_t^i) = \begin{cases} wait & \text{if } s_t^i \notin B(s^1, ..., s^N) \\ & \quad \text{and } \exists j : s_t^j \in B(s^1, ..., s^N) \\ wait & \text{if } s_t^i \in B(s^1, ..., s^N) \\ & \quad \text{and } \exists j : s_t^j \in B(s^1, ..., s^N) \\ & \quad \text{with } \beta^i(t) > \beta^j(t) \\ go & \text{else,} \end{cases}$$

where in the event of a tie between two or more agents for the minimum remaining time in the bottleneck, agent i is randomly generated and communicated between agents such that $P^i(o_0^i ... o_t^i) = go$ and $P^k(o_0^k ... o_t^k) = wait$ for all other agents k.

Proposition 4: Assume that for any agent i in the bottleneck, agent i at time t does not leave the bottleneck unless $P^i(o_0^i .. o_t^i) = wait$. Assume also that agent i passes through the bottleneck after β consecutive timesteps in the bottleneck. Then SRTF is a scheduling protocol for bottleneck $B(s^1, ..., s^N)$ with necessary time β.

Proof: As SRTF is a first come first serve protocol, we only consider the case when multiple agents arrive at the bottleneck at t. Let i be the agent with the shortest remaining time β^i to pass through the bottleneck. At time t, $P^i(o_0^i ... o_t^i) = go$ and $P^k(o_0^k ... o_t^k) = wait$ for all $k \neq i$. Then, since agent i does not leave the bottleneck until it passes through, for all $t' \in [t, t + \beta^i]$, $P^i(o_0^i ... o_{t'}^i) = go$ and $P^k(o_0^k ... o_{t'}^k) = wait$ for all $k \neq i$. If agent i has the shortest remaining time, agent i will receive a turn of length β. Since there are finite number of agents, the number of times an agent can be forced to leave the bottleneck is finite. Thus, at some t, agent i will arrive and have either the shortest time remaining or win the tie and receive a turn of length β. ■

For SRTF, local observations o^i at a minimum must include an agent's local state, the time remaining for the other agents in a bottleneck, and a signal for if the bottleneck is already occupied by another agent. This assumes that the agent can estimate the time remaining for other agents when they are in the bottleneck. Under the assumption that agents can communicate the outcome of a tie with other agents in a bottleneck, SRTF is a fully decentralized protocol. Alternatively, implementing exponential backoff in the event of ties allows for a fully decentralized protocol. Otherwise, a centralized authority is required to handle ties in SRTF.

SRTF has many of the same benefits as CSMA/CD, but it can be more efficient than CSMA/CD when collisions occur. One agent is allowed to continue using the resource rather than backing off. However, the ability to predict the shortest remaining time may not be a viable assumption in some environments.

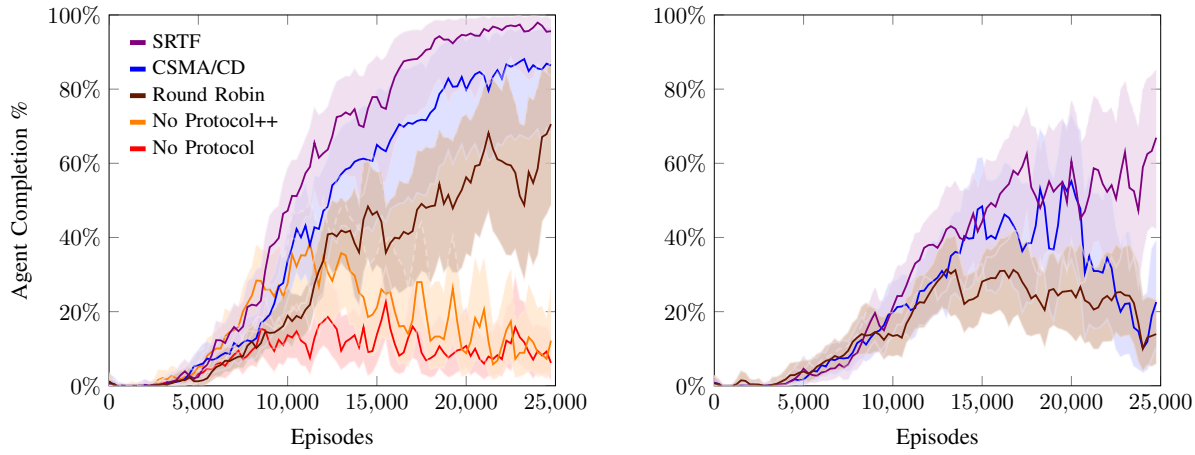

Fig. 4: Experimental Results for the hallway environment with 15 agents. Shaded areas indicate 25% and 75% quantiles over 5 seeds. Fig. 4a shows a comparison of agent task completion rates during decentralized, concurrent training. Fig. 4b shows an ablation where protocols affect the state space of each agent, but no reward penalty is applied for protocol violations.

D. MARL with Protocols

We present a decentralized, concurrent MARL training scheme for dec-MDPs with bottleneck resources. The approach uses a resource scheduling protocol for each bottleneck in the environment. The protocol's output for each agent is appended to that agent's local state. Agents learn a separate policy for each output of the protocol according to their local reward function. When the protocol signals an agent to wait, this agent is additionally punished via a reward penalty if it occupies a bottleneck state, violating the protocol.

We assume that a change in the protocol state between timesteps is caused by external factors, such as other agents entering the bottleneck resource, so we do not propagate value between protocol states for a given agent. As a result, each agent learns a separate policy for each state of the protocol as if that protocol state continues indefinitely. This ensures that an agent does not learn to avoid bottlenecks due to deadlocks that occur when another agent violates the protocol. See Fig. 3 for a diagram showing how π_{go} and π_{wait} are trained using a given trajectory.

Given the protocol state and the local state, an agent can learn a policy for each protocol state using any RL subroutine. While learning how to maximize their local reward function, agents also learn to comply with the protocol due to the reward penalty. Since complying with a protocol provably allows every agent to achieve high reward, the agents can collectively learn to achieve high total reward while only optimizing for local rewards.

V. Experiments

In this section we evaluate the performance of various scheduling protocols in a simulated multi-agent environment. We are primarily interested in the percentage of agents who complete their task during training. We also perform an ablation to evaluate the importance of penalizing protocol violations. The code can be found on GitHub: `https://github.com/tyler-ingebrand/SchedulingProtocolMARL`.

A. Simulation

We developed a multi-agent simulation environment to evaluate the scheduling protocols. The environment consists of a 10x10 grid with two rooms and a single hallway connecting them, as shown in Fig. 2. We randomly spawn 15 agents throughout both rooms, and each agent is tasked with traversing to a destination square in the other room. To increase the likelihood of a deadlock, we intentionally set the number of agents high relative to the available space. The state of each agent includes its own position and the state of the protocol, if applicable. The action space for each agent consists of four possible actions: move up, down, left, or right.

Agents are decentralized and trained concurrently to accomplish their task while following a shared scheduling protocol. Any RL subroutine could be used to learn a policy. For simplicity, we use a tabular Q-learning approach.

B. Baselines

We compare the performance under the three scheduling protocols to two vanilla MARL baselines described below, No Protocol and No Protocol++.

a) No Protocol: This baseline is a naïve decentralized approach. Each agent is only given its local state. Because agents are not aware of the presence of other agents in the environment, they cannot detect when a deadlock has occurred and are not penalized for participating in a deadlock.

b) No Protocol++: This baseline is similar to No Protocol, but adds a signal to the state space indicating when a deadlock has occurred. The agents' policy may leverage this information to resolve the deadlock. However, no penalty

is applied for participating in a deadlock, so the only reward incentives come from task completion.

C. Results

We evaluate the performance of the scheduling protocols based on the number of agents who complete their task at convergence. We find that the use of any scheduling protocol improves performance. See Fig. 4a. The scheduling protocols converge to a high success rate. In contrast, the MARL baselines show initial improvements but ultimately converge to poor performance due to unresolved deadlocks.

In an ablation, we evaluate the same protocols but without a penalty for violating the protocol. The goal of this ablation is to determine if the extra information provided by the protocol is sufficient to encourage cooperation, or if the reward penalty is needed. We refer the reader to Fig. 4b. RR and CSMA/CD converge towards poor performance without a reward penalty. SRTF is the only protocol which shows a stable agent completion rate, though it is significantly worse than the performance with a reward penalty. SRTF may therefore be a useful scheduling protocol where it is impractical or undesirable to apply a reward penalty.

VI. Conclusion

In this work we introduced the use of scheduling protocols to resolve conflicts in MARL. The use of scheduling protocols provably allows all agents to use shared resources without blocking one another. We have shown how decentralized MARL agents can be trained to follow a protocol by modifying their state space and applying a penalty for protocol violations. Experimental results demonstrate the effectiveness of three scheduling protocols in a crowded grid-world environment, with all three protocols outperforming MARL baselines.

References

[1] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. "Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications". In: *IEEE Transactions on Cybernetics* (2020). DOI: 10.1109/tcyb.2020.2977374.

[2] IEEE. "IEEE Standards for Local Area Networks: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications". In: *ANSI/IEEE Std 802.3-1985* (1985). DOI: 10.1109/IEEESTD.1985.82837.

[3] Harshal Bharatkumar Parekh and Sheetal Chaudhari. "Improved Round Robin CPU Scheduling Algorithm: Round Robin, Shortest Job First and Priority Algorithm Coupled to Increase Throughput and Decrease Waiting Time and Turnaround Time". In: *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*. 2016. DOI: 10.1109/ICGTSPICC.2016.7955294.

[4] Paulina Varshavskaya, Leslie Pack Kaelbling, and Daniela Rus. "Efficient Distributed Reinforcement Learning through Agreement". In: *Distributed Autonomous Robotic Systems 8*. 2008. DOI: 10.1007/978-3-642-00644-9_33.

[5] Paris Pennesi and Ioannis Ch. Paschalidis. "A Distributed Actor-Critic Algorithm and Applications to Mobile Sensor Network Coordination Problems". In: *IEEE Trans. Autom. Control.* (2010). DOI: 10.1109/TAC.2009.2037462.

[6] Kaiqing Zhang et al. "Fully Decentralized Multi-Agent Reinforcement Learning with Networked Agents". In: *CoRR* abs/1802.08757 (2018).

[7] Francisco Melo and Manuela Veloso. "Learning of Coordination: Exploiting Sparse Interactions in Multiagent Systems." In: 2009. DOI: 10.1145/1558109.1558118.

[8] Soummya Kar, José M. F. Moura, and H. Vincent Poor. "$\mathcal{QD}$-Learning: A Collaborative Distributed Strategy for Multi-Agent Reinforcement Learning Through $\mathrm{Consensus} + \mathrm{Innovations}$". In: *IEEE Transactions on Signal Processing* 61.7 (2013), pp. 1848–1862. DOI: 10.1109/tsp.2013.2241057.

[9] Yuandong Ding et al. *Multi-Agent Reinforcement Learning with Shared Resources for Inventory Management*. 2022. DOI: 10.48550/ARXIV.2212.07684.

[10] Ingy Elsayed-Aly et al. "Safe Multi-Agent Reinforcement Learning via Shielding". In: *CoRR* abs/2101.11196 (2021).

[11] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. DOI: https://doi.org/10.48550/arXiv.1312.5602.

[12] Timothy P. Lillicrap et al. *Continuous Control with Deep Reinforcement Learning*. 2019. DOI: https://doi.org/10.48550/arXiv.1509.02971.

[13] Matteo Hessel et al. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. 2017. DOI: https://doi.org/10.48550/arXiv.1710.02298.

[14] Julian Schrittwieser et al. "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model". In: *Nature* (2020). DOI: 10.1038/s41586-020-03051-4.

[15] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. DOI: https://doi.org/10.48550/arXiv.1707.06347.

[16] R. Becker, S. Zilberstein, and V. Lesser. "Decentralized Markov Decision Processes with Event-Driven Interactions". In: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*. 2004, pp. 302–309.

[17] Ming Tan. "Multi-Agent Reinforcement Learning: Independent versus Cooperative Agents". In: *Proceedings of the Tenth International Conference on Machine Learning (ICML 1993)*. 1993, pp. 330–337.