Check for updates

CAV Artifact Evaluation ★★ Functional

Specification-Guided Learning of Nash Equilibria with High Social Welfare

Kishor Jothimurugan(✉), Suguman Bansal, Osbert Bastani, and Rajeev Alur

University of Pennsylvania, Philadelphia, USA
kishor@seas.upenn.edu

Abstract. Reinforcement learning has been shown to be an effective strategy for automatically training policies for challenging control problems. Focusing on non-cooperative multi-agent systems, we propose a novel reinforcement learning framework for training joint policies that form a Nash equilibrium. In our approach, rather than providing low-level reward functions, the user provides high-level specifications that encode the objective of each agent. Then, guided by the structure of the specifications, our algorithm searches over policies to identify one that provably forms an ϵ-Nash equilibrium (with high probability). Importantly, it prioritizes policies in a way that maximizes social welfare across all agents. Our empirical evaluation demonstrates that our algorithm computes equilibrium policies with high social welfare, whereas state-of-the-art baselines either fail to compute Nash equilibria or compute ones with comparatively lower social welfare.

1 Introduction

Reinforcement learning (RL) is an effective strategy for automatically synthesizing controllers for challenging control problems. As a consequence, there has been interest in applying RL to multi-agent systems. For example, RL has been used to coordinate agents in cooperative systems to accomplish a shared goal [22]. Our focus is on non-cooperative systems, where the agents are trying to achieve their own goals [17]; for such systems, the goal is typically to learn a policy for each agent such that the joint strategy forms a Nash equilibrium.

A key challenge facing existing approaches is how tasks are specified. First, they typically require that the task for each agent is specified as a reward function. However, reward functions tend to be very low-level, making them difficult to manually design; furthermore, they often obfuscate high-level structure in the problem known to make RL more efficient in the single-agent [14] and cooperative [22] settings. Second, they typically focus on computing an arbitrary Nash equilibrium. However, in many settings, the user is a social planner trying to optimize the overall social welfare of the system, and most existing approaches are not designed to optimize social welfare.

We propose a novel multi-agent RL framework for learning policies from high-level specifications (one specification per agent) such that the resulting

The extended version of this paper can be found at [3].

© The Author(s) 2022
S. Shoham and Y. Vizel (Eds.): CAV 2022, LNCS 13372, pp. 343–363, 2022.
https://doi.org/10.1007/978-3-031-13188-2_17

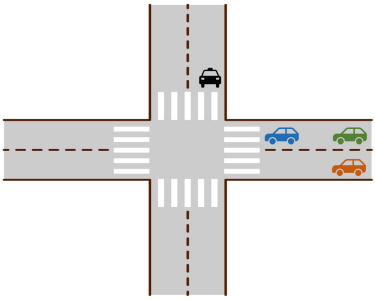

Fig. 1. Intersection Example

joint policy (i) has high social welfare, and (ii) is an ϵ-Nash equilibrium (for a given ϵ). We formulate this problem as a constrained optimization problem where the goal is to maximize social welfare under the constraint that the joint policy is an ϵ-Nash equilibrium.

Our algorithm for solving this optimization problem uses an enumerative search strategy. First, it enumerates candidate policies in decreasing order of social welfare. To ensure a tractable search space, it restricts to policies that conform to the structure of the user-provided specification. Then, for each candidate policy, it uses an explore-then-exploit self-play RL algorithm [4] to compute *punishment strategies* that are triggered when some agent deviates from the original joint policy. It also computes the maximum benefit each agent derives from deviating, which can be used to determine whether the joint policy augmented with punishment strategies forms an ϵ-Nash equilibrium; if so, it returns the joint policy.

Intuitively, the enumerative search tries to optimize social welfare, whereas the self-play RL algorithm checks whether the ϵ-Nash equilibrium constraint holds. Since this RL algorithm comes with PAC (Probably Approximately Correct) guarantees, our algorithm is guaranteed to return an ϵ-Nash equilibrium with high probability. In summary, our contributions are as follows.

- We study the problem of maximizing social welfare under the constraint that the policies form an ϵ-NE. To the best of our knowledge, this problem has not been studied before in the context of learning (beyond single-step games).
- We provide an enumerate-and-verify framework for solving the said problem.
- We propose a verification algorithm with a probabilistic soundness guarantee in the RL setting of probabilistic systems with unknown transition probabilities.

Motivating Example. Consider the road intersection scenario in Fig. 1. There are four cars; three are traveling east to west and one is traveling north to south. At any stage, each car can either move forward one step or stay in place. Suppose each car's specification is as follows:

- *Black car:* Cross the intersection before the green and orange cars.
- *Blue car:* Cross the intersection before the black car and stay a car length ahead of the green and orange cars.
- *Green car:* Cross the intersection before the black car.
- *Orange car:* Cross the intersection before the black car.

We also require that the cars do not crash into one another.

Clearly, not all agents can achieve their goals. The next highest social welfare is for three agents to achieve their goals. In particular, one possibility is that all cars except the black car achieve their goals. However, the corresponding joint policy requires that the black car does not move, which is not a Nash equilibrium—there is always a gap between the blue car and the other two cars behind, so the black car can deviate by inserting itself into the gap to achieve its own goal. Our algorithm uses self-play RL to optimize the policy for the black car, and finds that the other agents cannot prevent the black car from improving its outcome in this way. Thus, it correctly rejects this joint policy. Eventually, our algorithm computes a Nash equilibrium in which the black and blue cars achieve their goals.

1.1 Related Work

Multi-agent RL. There has been work on learning Nash equilibria in the multi-agent RL setting [1,12,13,21,23,24]; however, these approaches focus on learning an arbitrary equilibrium and do not optimize social welfare. There has also been work on studying weaker notions of equilibria in this context [9,27], as well as work on learning Nash equilibria in two agent zero-sum games [4,20,26].

RL from High-Level Specifications. There has been recent work on using specifications based on temporal logic for specifying RL tasks in the single agent setting; a comprehensive survey may be found in [2]. There has also been recent work on using temporal logic specifications for multi-agent RL [10,22], but these approaches focus on cooperative scenarios in which there is a common objective that all agents are trying to achieve.

Equilibrium in Markov Games. There has been work on computing Nash equilibrium in Markov games [17,25], including work on computing ϵ-Nash equilibria from logical specifications [6,7], as well as recent work focusing on computing welfare-optimizing Nash equilibria from temporal specifications [18,19]; however, all these works focus on the planning setting where the transition probabilities are known. Checking for existence of Nash equilibrium, even in deterministic games, has been shown to be NP-complete for reachability objectives [5].

Social Welfare. There has been work on computing welfare maximizing Nash equilibria for bimatrix games, which are two-player one-step Markov games with known transitions [8,11]; in contrast, we study this problem in the context of general Markov games.

2 Preliminaries

2.1 Markov Game

We consider an n-agent Markov game $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, H, s_0)$ with a finite set of states $\mathcal{S}$, actions $\mathcal{A} = A_1 \times \cdots \times A_n$ where A_i is a finite set of actions available to agent i, transition probabilities $P(s' \mid s, a)$ for $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$, finite horizon H, and initial state s_0 [20]. A *trajectory* $\zeta \in \mathcal{Z} = (\mathcal{S} \times \mathcal{A})^* \times \mathcal{S}$ is a finite sequence $\zeta = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{t-1}} s_t$ where $s_k \in \mathcal{S}$, $a_k \in \mathcal{A}$; we use $|\zeta| = t$ to denote the length of the trajectory ζ and $a_k^i \in A_i$ to denote the action of agent i in a_k.

For any $i \in [n]$, let $\mathcal{D}(A_i)$ denote the set of distributions over A_i—i.e., $\mathcal{D}(A_i) = \{\Delta : A_i \to [0,1] \mid \sum_{a_i \in A_i} \Delta(a_i) = 1\}$. A *policy* for agent i is a function $\pi_i : \mathcal{Z} \to \mathcal{D}(A_i)$ mapping trajectories to distributions over actions. A policy π_i is *deterministic* if for every $\zeta \in \mathcal{Z}$, there is an action $a_i \in A_i$ such that $\pi_i(\zeta)(a_i) = 1$; in this case, we also use $\pi_i(\zeta)$ to denote the action a_i. A *joint policy* $\pi : \mathcal{Z} \to \mathcal{D}(A)$ maps finite trajectories to distributions over joint actions. We use $(\pi_1, \ldots, \pi_n)$ to denote the joint policy in which agent i chooses its action in accordance to π_i. We denote by $\mathcal{D}_\pi$ the distribution over H-length trajectories in $\mathcal{M}$ induced by π.

We consider the reinforcement learning setting in which we do not know the probabilities P but instead only have access to a simulator of $\mathcal{M}$. Typically, we can only sample trajectories of $\mathcal{M}$ starting at s_0. Some parts of our algorithm are based on an assumption which allows us to obtain sample trajectories starting at any state that has been observed before. For example, if taking action a_0 in s_0 leads to a state s_1, we assume we can obtain future samples starting at s_1.

Assumption 1. *We can obtain samples from $P(\cdot \mid s, a)$ for any previously observed state s and any action a.*

2.2 Specification Language

We consider the specification language Spectrl to express agent specifications. We choose Spectrl since there is existing work on leveraging the structure of Spectrl specifications for single-agent RL [16]. However, we believe our algorithm can be adapted to other specification languages as well.

Formally, a Spectrl specification is defined over a set of *atomic predicates* $\mathcal{P}_0$, where every $p \in \mathcal{P}_0$ is associated with a function $[\![p]\!] : \mathcal{S} \to \mathbb{B} = \{\texttt{true}, \texttt{false}\}$; we say a state s *satisfies* p (denoted $s \models p$) if and only if $[\![p]\!](s) = \texttt{true}$. The set of *predicates* $\mathcal{P}$ consists of conjunctions and disjunctions of atomic predicates. The syntax of a predicate $b \in \mathcal{P}$ is given by the grammar $b ::= p \mid (b_1 \wedge b_2) \mid (b_1 \vee b_2)$, where $p \in \mathcal{P}_0$. Similar to atomic predicates, each

predicate $b \in \mathcal{P}$ corresponds to a function $[\![b]\!] : \mathcal{S} \to \mathbb{B}$ defined naturally over Boolean logic. Finally, the syntax of SPECTRL is given by[1]

$$\phi \ ::= \ \texttt{achieve}\ b \mid \phi_1\ \texttt{ensuring}\ b \mid \phi_1; \phi_2 \mid \phi_1\ \texttt{or}\ \phi_2,$$

where $b \in \mathcal{P}$. Each specification ϕ corresponds to a function $[\![\phi]\!] : \mathcal{Z} \to \mathbb{B}$, and we say $\zeta \in \mathcal{Z}$ satisfies ϕ (denoted $\zeta \models \phi$) if and only if $[\![\phi]\!](\zeta) = \texttt{true}$. Letting ζ be a finite trajectory of length t, this function is defined by

$$\begin{aligned}
&\zeta \models \texttt{achieve}\ b && \text{if } \exists\ i \le t,\ s_i \models b \\
&\zeta \models \phi\ \texttt{ensuring}\ b && \text{if } \zeta \models \phi \text{ and } \forall\ i \le t,\ s_i \models b \\
&\zeta \models \phi_1; \phi_2 && \text{if } \exists\ i < t,\ \zeta_{0:i} \models \phi_1 \text{ and } \zeta_{i+1:t} \models \phi_2 \\
&\zeta \models \phi_1\ \texttt{or}\ \phi_2 && \text{if } \zeta \models \phi_1 \text{ or } \zeta \models \phi_2.
\end{aligned}$$

Intuitively, the first clause means that the trajectory should eventually reach a state that satisfies the predicate b. The second clause says that the trajectory should satisfy specification ϕ while always staying in states that satisfy b. The third clause says that the trajectory should sequentially satisfy ϕ_1 followed by ϕ_2. The fourth clause means that the trajectory should satisfy either ϕ_1 or ϕ_2.

2.3 Abstract Graphs

SPECTRL specifications can be represented by *abstract graphs* which are DAG-like structures in which each vertex represents a set of states (called subgoal regions) and each edge represents a set of concrete trajectories that can be used to transition from the source vertex to the target vertex without violating safety constraints.

Definition 1. An *abstract graph* $\mathcal{G} = (U, E, u_0, F, \beta, \mathcal{Z}_{\text{safe}})$ is a directed acyclic graph (DAG) with vertices U, (directed) edges $E \subseteq U \times U$, initial vertex $u_0 \in U$, final vertices $F \subseteq U$, subgoal region map $\beta : U \to 2^S$ such that for each $u \in U$, $\beta(u)$ is a subgoal region,[2] and *safe trajectories* $\mathcal{Z}_{\text{safe}} = \bigcup_{e \in E} \mathcal{Z}^e_{\text{safe}} \cup \bigcup_{f \in F} \mathcal{Z}^f_{\text{safe}}$, where $\mathcal{Z}^e_{\text{safe}} \subseteq \mathcal{Z}$ denotes the safe trajectories for edge $e \in E$ and $\mathcal{Z}^f_{\text{safe}} \subseteq \mathcal{Z}$ denotes the safe trajectories for final vertex $f \in F$.

Intuitively, (U, E) is a standard DAG, and u_0 and F define a graph reachability problem for (U, E). Furthermore, β and $\mathcal{Z}_{\text{safe}}$ connect (U, E) back to the original MDP $\mathcal{M}$; in particular, for an edge $e = u \to u'$, $\mathcal{Z}^e_{\text{safe}}$ is the set of safe trajectories in $\mathcal{M}$ that can be used to transition from $\beta(u)$ to $\beta(u')$.

Definition 2. A trajectory $\zeta = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{t-1}} s_t$ in $\mathcal{M}$ satisfies the abstract graph $\mathcal{G}$ (denoted $\zeta \models \mathcal{G}$) if there is a sequence of indices $0 = k_0 \le k_1 < \cdots < k_\ell \le t$ and a path $\rho = u_0 \to u_1 \to \cdots \to u_\ell$ in $\mathcal{G}$ such that

[1] Here, `achieve` and `ensuring` correspond to the "eventually" and "always" operators in temporal logic.

[2] We do not require that the subgoal regions partition the state space or that they be non-overlapping.

- $u_\ell \in F$,
- for all $z \in \{0, \ldots, \ell\}$, we have $s_{k_z} \in \beta(u_z)$,
- for all $z < \ell$, letting $e_z = u_z \to u_{z+1}$, we have $\zeta_{k_z:k_{z+1}} \in \mathcal{Z}_{\text{safe}}^{e_z}$, and
- $\zeta_{k_\ell:t} \in \mathcal{Z}_{\text{safe}}^{u_\ell}$.

The first two conditions state that the trajectory should visit a sequence of subgoal regions corresponding to a path from the initial vertex to some final vertex, and the last two conditions state that the trajectory should be composed of subtrajectories that are safe according to $\mathcal{Z}_{\text{safe}}$.

Prior work shows that for every SPECTRL specification ϕ, we can construct an abstract graph $\mathcal{G}_\phi$ such that for every trajectory $\zeta \in \mathcal{Z}$, $\zeta \models \phi$ if and only if $\zeta \models \mathcal{G}_\phi$ [16]. Finally, the number of states in the abstract graph is linear in the size of the specification.

2.4 Nash Equilibrium and Social Welfare

Given a Markov game $\mathcal{M}$ with unknown transitions and SPECTRL specifications $\phi_1, \ldots, \phi_n$ for the n agents respectively, the score of agent i from a joint policy π is given by

$$J_i(\pi) = \Pr_{\zeta \sim \mathcal{D}_\pi}[\zeta \models \phi_i].$$

Our goal is to compute a *high-value* ϵ-Nash equilibrium in $\mathcal{M}$ w.r.t these scores. Given a joint policy $\pi = (\pi_1, \ldots, \pi_n)$ and an alternate policy π_i' for agent i, let (π_{-i}, π_i') denote the joint policy $(\pi_1, \ldots, \pi_i', \ldots, \pi_n)$. Then, a joint policy π is an *ϵ-Nash equilibrium* if for all agents i and all alternate policies π_i', $J_i(\pi) \geq J_i((\pi_{-i}, \pi_i')) - \epsilon$. Our goal is to compute a joint policy π that maximizes the social welfare given by

$$\texttt{welfare}(\pi) = \frac{1}{n} \sum_{i=1}^{n} J_i(\pi)$$

subject to the constraint that π is an ϵ-Nash equilibrium.

3 Overview

Our framework for computing a high-welfare ϵ-Nash equilibrium consists of two phases. The first phase is a *prioritized enumeration* procedure that learns deterministic joint policies in the environment and ranks them in decreasing order of social welfare. The second phase is a *verification phase* that checks whether a given joint policy can be extended to an ϵ-Nash equilibrium by adding punishment strategies. A policy is returned if it passes the verification check in the second phase. Algorithm 1 summarizes our framework.

For the enumeration phase, it is impractical to enumerate all joint policies even for small environments, since the total number of deterministic joint policies is $\Omega(|\mathcal{A}|^{|\mathcal{S}|^{H-1}})$, which is $\Omega(2^{n|\mathcal{S}|^{H-1}})$ if each agent has atleast two actions. Thus,

Algorithm 1 HighNashSearch
Inputs: Markov game (with unknown transition probabilities) $\mathcal{M}$ with n-agents, agent specifications $\phi_1, \ldots, \phi_n$, Nash factor ϵ, precision δ, failure probability p.
Outputs: ϵ-NE, if found.

1: PrioritizedPolicies $\leftarrow$ PrioritizedEnumeration$(\mathcal{M}, \phi_1, \ldots, \phi_n)$
2: **for** joint policy $\pi \in$ PrioritizedPolicies **do**
3: // Can π be extended to an ϵ-NE?
4: `isNash`, $\tau \leftarrow$ VerifyNash$(\mathcal{M}, \pi, \phi_1, \cdots, \phi_n, \epsilon, \delta, p)$
5: **if** `isNash` **then return** $\pi \bowtie \tau$ // Add punishment strategies
6: **return** No ϵ-NE found

in the prioritized enumeration phase, we apply a specification-guided heuristic to reduce the number of joint policies considered. The resulting search space is independent of $|\mathcal{S}|$ and H, depending only on the specifications $\{\phi_i\}_{i\in[n]}$. Since the transition probabilities are unknown, these joint policies are trained using an efficient compositional RL approach.

Since the joint policies are trained cooperatively, they are typically not ϵ-Nash equilibria. Hence, in the verification phase, we use a probably approximately correct (PAC) procedure (Algorithm 2) to determine whether a given joint policy can be modified by adding *punishment strategies* to form an ϵ-Nash equilibrium. Our approach is to reduce this problem to solving two-agent zero-sum games. The key insight is that for a given joint policy to be an ϵ-Nash equilibrium, unilateral deviations by any agent must be successfully punished by the coalition of all other agents. In such a *punishment game*, the deviating agent attempts to maximize its score while the coalition of other agents attempts to minimize its score, leading to a competitive min-max game between the agent and the coalition. If the deviating agent can improve its score by a margin $\geq \epsilon$, then the joint policy cannot be extended to an ϵ-Nash equilibrium. Alternatively, if no agent can increase its score by a margin $\geq \epsilon$, then the joint policy (augmented with punishment strategies) is an ϵ-Nash equilibrium. Thus, checking if a joint policy can be converted to an ϵ-Nash equilibrium reduces to solving a two-agent zero-sum game for each agent. Each punishment game is solved using a self-play RL algorithm for learning policies in min-max games with unknown transitions [4], after converting specification-based scores to reward-based scores. While the initial joint policy is deterministic, the punishment strategies can be probabilistic.

Overall, we provide the guarantee that with high probability, if our algorithm returns a joint policy, it will be an ϵ-Nash equilibrium.

4 Prioritized Enumeration

We summarize our specification-guided compositional RL algorithm for learning a finite number of deterministic joint policies in an unknown environment under Assumption 1. These policies are then ranked in decreasing order of their (estimated) social welfare.

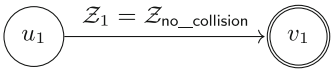

Fig. 2. Abstract Graph of black car.

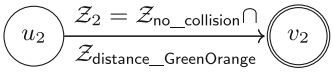

Fig. 3. Abstract Graph of blue car.

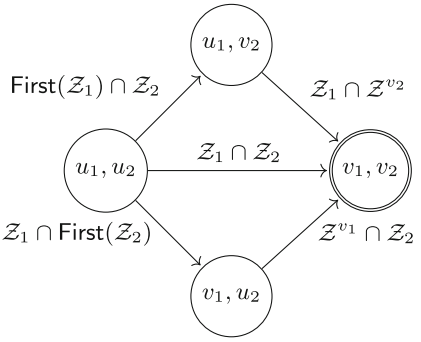

Fig. 4. Product Abstract Graph of black and blue cars. $\mathcal{Z}^{v_1}$ and $\mathcal{Z}^{v_2}$ refer to safe trajectories after the black and blue cars have reached their final states, respectively.

Our learning algorithm harnesses the structure of specifications, exposed by their abstract graphs, to curb the number of joint policies to learn. For every set of *active agents* $B \subseteq [n]$, we construct a product abstract graph, from the abstract graphs of all active agents' specifications. A property of this product is that if a trajectory ζ in $\mathcal{M}$ corresponds to a path in the product that ends in a final state then ζ satisfies the specification of *all* active agents. Then, our procedure learns one joint policy for every path in the product graph that reaches a final state. Intuitively, policies learned using the product graph corresponding to a set of active agents B aim to maximize satisfaction probabilities of all agents in B. By learning joint policies for every set of active agents, we are able to learn policies under which some agents may not satisfy their specifications. This enables learning joint policies in non-cooperative settings. Note that the number of paths (and hence the number of policies considered) is independent of $|\mathcal{S}|$ and H, and depends only on the number of agents and their specifications.

One caveat is that the number of paths may be exponential in the number of states in the product graph. It would be impractical to naïvely learn a joint policy for every path. Instead, we design an efficient compositional RL algorithm that learns a joint policy for each edge in the product graph; these edge policies are then composed together to obtain joint policies for paths in the product graph.

4.1 Product Abstract Graph

Let $\phi_1, \ldots, \phi_n$ be the specifications for the n-agents, respectively, and let $\mathcal{G}_i = (U_i, E_i, u_0^i, F_i, \beta_i, \overline{\mathcal{Z}}_{\text{safe},i})$ be the abstract graph of specification ϕ_i in the environment $\mathcal{M}$. We construct a product abstract graph for every set of active agents in $[n]$. The product graph for a set of active agents $B \subseteq [n]$ is used to

learn joint policies which satisfy the specification of all agents in B with high probability.

Definition 3. *Given a set of agents $B = \{i_1, \ldots, i_m\} \subseteq [n]$, the product graph $\mathcal{G}_B = (\overline{U}, \overline{E}, \overline{u}_0, \overline{F}, \overline{\beta}, \overline{\mathcal{Z}}_{\text{safe}})$ is the asynchronous product of $\mathcal{G}_i$ for all $i \in B$, with*

- *$\overline{U} = \prod_{i \in B} U_i$ is the set of product vertices,*
- *An edge $e = (u_{i_1}, \ldots, u_{i_m}) \to (v_{i_1}, \ldots, v_{i_m}) \in \overline{E}$ if at least for one agent $i \in B$ the edge $u_i \to v_i \in E_i$ and for the remaining agents, $u_i = v_i$,*
- *$\overline{u}_0 = (u_0^{i_1}, \ldots, u_0^{i_m})$ is the initial vertex,*
- *$\overline{F} = \Pi_{i \in B} F_i$ is the set of final vertices,*
- *$\overline{\beta} = (\beta_{i_1}, \ldots, \beta_{i_m})$ is the collection of concretization maps, and*
- *$\overline{\mathcal{Z}}_{\text{safe}} = (\overline{\mathcal{Z}}_{\text{safe},i_1}, \ldots, \overline{\mathcal{Z}}_{\text{safe},i_m})$ is the collection of safe trajectories.*

We denote the i-th component of a product vertex $\overline{u} \in \overline{U}$ by u_i for agent $i \in B$. Similarly, the i-th component in an edge $e = \overline{u} \to \overline{v}$ is denoted by $e_i = u_i \to v_i$ for $i \in B$; note that e_i can be a self loop which is not an edge in $\mathcal{G}_i$. For an edge $e \in \overline{E}$, we denote the set of agents $i \in B$ for which $e_i \in E_i$, and not a self loop, by $\mathsf{progress}(e)$.

Abstract graphs of the black car and the blue car from the motivating example are shown in Figs. 2 and 3 respectively. The vertex v_1 denotes the subgoal region $\beta_{\text{black}}(v_1)$ consisting of states in which the black car has crossed the intersection but the orange and green cars have not. The subgoal region $\beta_{\text{blue}}(v_2)$ is the set of states in which the blue car has crossed the intersection but the black car has not. $\mathcal{Z}_1$ denotes trajectories in which the black car does not collide and $\mathcal{Z}_2$ denotes trajectories in which the blue car does not collide and stays a car length ahead of the orange and green cars. The product abstract graph for the set of active agents $B = \{\text{black, blue}\}$ is shown in Fig 4. The safe trajectories on the edges reflect the notion of *achieving* a product edge which we discuss below.

A trajectory $\zeta = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{t-1}} s_t$ *achieves* an edge $e = \overline{u} \to \overline{v}$ in $\mathcal{G}_B$ if all progressing agents $i \in \mathsf{progress}(e)$ reach their target subgoal region $\beta_i(v_i)$ along the trajectory and the trajectory is safe for all agents in B. For a progressing agent $i \in \mathsf{progress}(e)$, the initial segment of the rollout until the agent reaches its subgoal region should be safe with respect to the edge e_i. After that, the rollout should be safe with respect to every future possibility for the agent. This is required to ensure continuity of the rollout into adjacent edges in the product graph $\mathcal{G}_B$. For the same reason, we require that the entire rollout is safe with respect to all future possibilities for non-progressing agents. Note that we are not concerned with non-active agents in $[n] \backslash B$. In order to formally define this notion, we need to setup some notation.

For a predicate $b \in \mathcal{P}$, let the set of safe trajectories w.r.t. b be given by $\mathcal{Z}_b = \{\zeta = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{t-1}} s_t \in \mathcal{Z} \mid \forall\ 0 \leq k \leq t, s_k \models b\}$. It is known that safe trajectories along an edge in an abstract graph constructed from a SPECTRL specification is either of the form $\mathcal{Z}_b$ or $\mathcal{Z}_{b_1} \circ \mathcal{Z}_{b_2}$, where $b, b_1, b_2 \in \mathcal{P}$ and $\circ$ denotes concatenation [16]. In addition, for every final vertex f, $\mathcal{Z}^f_{\text{safe}}$ is of the form $\mathcal{Z}_b$ for some $b \in \mathcal{P}$. We define First as follows:

$$\mathsf{First}(\mathcal{Z}') = \begin{cases} \mathcal{Z}_b, & \text{if } \mathcal{Z}' = \mathcal{Z}_b \\ \mathcal{Z}_{b_1}, & \text{if } \mathcal{Z}' = \mathcal{Z}_{b_1} \circ \mathcal{Z}_{b_2} \end{cases}$$

We are now ready to define the notion of satisfiability of a product edge.

Definition 4. A rollout $\zeta = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{t-1}} s_k$ *achieves an edge* $e = \overline{u} \to \overline{v}$ in $\mathcal{G}_B$ (denoted $\zeta \models_B e$) if

1. for all progressing agents $i \in \mathsf{progress}(e)$, there exists an index $k_i \leq k$ such that $s_{k_i} \in \beta_i(v_i)$ and $\zeta_{0:k_i} \in \mathcal{Z}^{e_i}_{\text{safe},i}$. If $v_i \in F_i$ then $\zeta_{k_i:k} \in \mathcal{Z}^{v_i}_{\text{safe},i}$. Otherwise, $\zeta_{k_i:k} \in \mathsf{First}(\mathcal{Z}^{v_i \to w_i}_{\text{safe},i})$ for all $w_i \in \mathsf{outgoing}(v_i)$. Furthermore, we require $k_i > 0$ if $u_i \neq u^i_0$.
2. for all non-progressing agents $i \in B \backslash \mathsf{progress}(e)$, if $u_i \notin F_i$, $\zeta \in \mathsf{First}(\mathcal{Z}^{u_i \to w_i}_{\text{safe},i})$ for all $w_i \in \mathsf{outgoing}(u_i)$. Otherwise (if $u_i \in F_i$), $\zeta \in \mathcal{Z}^{u_i}_{\text{safe},i}$

We can now define what it means for a trajectory to achieve a path in the product graph $\mathcal{G}_B$.

Definition 5. Given $B \subseteq [n]$, a rollout $\zeta = s_0 \to \cdots \to s_t$ *achieves* a path $\rho = \overline{u}_0 \to \cdots \to \overline{u}_\ell$ in $\mathcal{G}_B$ (denoted $\zeta \models_B \rho$) if there exists indices $0 = k_0 \leq k_1 \leq \cdots \leq k_\ell \leq t$ such that (i) $\overline{u}_\ell \in \overline{F}$, (ii) $\zeta_{k_z:k_{z+1}}$ achieves $\overline{u}_z \to \overline{u}_{z+1}$ for all $0 \leq z < \ell$, and (iii) $\zeta_{k_\ell:t} \in \mathcal{Z}^{u_{\ell,i}}_{\text{safe},i}$ for all $i \in B$.

Theorem 2. *Let* $\rho = \overline{u_0} \to \overline{u_1} \to \cdots \to \overline{u_\ell}$ *be a path in the product abstract graph* $\mathcal{G}_B$ *for* $B \subseteq [n]$*. Suppose trajectory* $\zeta \models_B \rho$*. Then* $\zeta \models \phi_i$ *for all* $i \in B$*.*

That is, joint policies that maximize the probability of achieving paths in the product abstract graph $\mathcal{G}_B$ have high social welfare w.r.t. the active agents B.

4.2 Compositional RL Algorithm

Our compositional RL algorithm learns joint policies corresponding to paths in product abstract graphs. For every $B \subseteq [n]$, it learns a joint policy π_e for each edge in the product abstract graph $\mathcal{G}_B$, which is the (deterministic) policy that maximizes the probability of achieving e from a given initial state distribution. We assume all agents are acting cooperatively; thus, we treat the agents as one and use single-agent RL to learn each edge policy. We will check whether any deviation to this co-operative behaviour by any agent can be punished by the coalition of other agents in the verification phase. The reward function is designed to capture the reachability objective of progressing agents and the safety objective of all active agents.

The edges are learned in topological order, allowing us to learn an induced state distribution for each product vertex $\overline{u}$ prior to learning any edge policies from $\overline{u}$; this distribution is used as the initial state distribution when learning outgoing edge policies from $\overline{u}$. In more detail, the distribution for the initial vertex of $\mathcal{G}_B$ is taken to be the initial state distribution of the environment; for every other product vertex, the distribution is the average over distributions

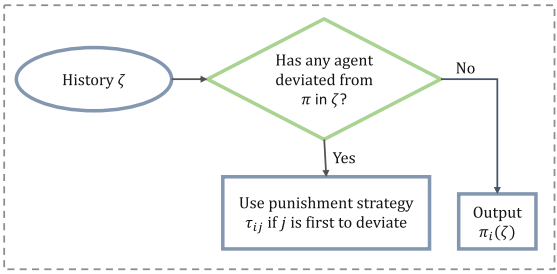

Fig. 5. π_i augmented with punishment strategies.

induced by executing edge policies for all incoming edges. This is possible because the product graph is a DAG.

Given edge policies Π along with a path $\rho = \overline{u}_0 \to \overline{u}_1 \to \cdots \to \overline{u}_\ell = \overline{u} \in \overline{F}$ in $\mathcal{G}_B$, we define a *path policy* π_ρ to navigate from $\overline{u}_0$ to $\overline{u}$. In particular, π_ρ executes $\pi_{e[z]}$, where $e[z] = \overline{u}_z \to \overline{u}_{z+1}$ (starting from $z = 0$) until the resulting trajectory achieves $e[z]$, after which it increments $z \leftarrow z + 1$ (unless $z = \ell$). That is, π_ρ is designed to achieve the sequence of edges in ρ. Note that π_ρ is a finite-state deterministic joint policy in which vertices on the path correspond to the memory states that keep track of the index of the current policy. This way, we obtain finite-state joint policies by learning edge policies only.

This process is repeated for all sets of active agents $B \subseteq [n]$. These finite-state joint policies are then ranked by estimating their social welfare on several simulations.

5 Nash Equilibria Verification

The prioritized enumearation phase produces a list of path policies which are ranked by the total sum of scores. Each path policy is deterministic and also finite state. Since the joint policies are trained cooperatively, they are typically not ϵ-Nash equilibria. Thus, our verification algorithm not only tries to prove that a given joint policy is a ϵ-Nash equilibrium, but also tries to modify it so it satisfies this property. In particular, our verification algorithm attempts to modify a given joint policy by adding *punishment strategies* so that the resulting policy is an ϵ-Nash equilibrium.

Concretely, it takes as input a finite-state deterministic joint policy $\pi = (M, \alpha, \sigma, m_0)$ where M is a finite set of *memory states*, $\alpha : \mathcal{S} \times \mathcal{A} \times M \to M$ is the memory update function, $\sigma : \mathcal{S} \times M \to \mathcal{A}$ maps states to (joint) actions and m_0 is the initial policy state. The *extended memory update function* $\hat{\alpha} : \mathcal{Z} \to M$ is given by $\hat{\alpha}(\epsilon) = m_0$ and $\hat{\alpha}(\zeta s_t a_t) = \alpha(s_t, a_t, \hat{\alpha}(\zeta))$. Then, π is given by $\pi(\zeta s_t) = \sigma(s_t, \hat{\alpha}(\zeta))$. The policy π_i of agent i simply chooses the i^{th} component of $\pi(\zeta)$ for any history ζ.

The verification algorithm learns one punishment strategy $\tau_{ij} : \mathcal{Z} \to \mathcal{D}(A_i)$ for each pair (i, j) of agents. As outlined in Fig. 5, the modified policy for agent i uses π_i if every agent j has taken actions according to π_j in the past. In case some agent j' has taken an action that does not match the output of $\pi_{j'}$, then agent i uses the punishment strategy τ_{ij}, where j is the agent that deviated the earliest (ties broken arbitrarily). The goal of verification is to check if there is a set of punishment strategies $\{\tau_{ij} \mid i \neq j\}$ such that after modifying each agent's policy to use them, the resulting joint policy is an ϵ-Nash equilibrium.

5.1 Problem Formulation

We denote the set of all punishment strategies of agent i by $\tau_i = \{\tau_{ij} \mid j \neq i\}$. We define the composition of π_i and τ_i to be the policy $\tilde{\pi}_i = \pi_i \bowtie \tau_i$ such that for any trajectory $\zeta = s_0 \xrightarrow{a_0} \cdots \xrightarrow{a_{t-1}} s_t$, we have

- $\tilde{\pi}_i(\zeta) = \pi_i(\zeta)$ if for all $0 \leq k < t$, $a_k = \pi(\zeta_{0:k})$—i.e., no agent has deviated so far,
- $\tilde{\pi}_i(\zeta) = \tau_{ij}(\zeta)$ if there is a k such that (i) $a_k^j \neq \pi_j(\zeta_{0:k})$ and (ii) for all $\ell < k$, $a_\ell = \pi(\zeta_{0:\ell})$. If there are multiple such j's, an arbitrary but consistent choice is made (e.g., the smallest such j).

Given a finite-state deterministic joint policy π, the verification problem is to check if there exists a set of punishment strategies $\tau = \bigcup_i \tau_i$ such that the joint policy $\tilde{\pi} = \pi \bowtie \tau = (\pi_1 \bowtie \tau_1, \ldots, \pi_n \bowtie \tau_n)$ is an ϵ-Nash equilibrium. In other words, the problem is to check if there exists a policy $\tilde{\pi}_i$ for each agent i such that (i) $\tilde{\pi}_i$ follows π_i as long as no other agent j deviates from π_j and (ii) the joint policy $\tilde{\pi} = (\tilde{\pi}_1, \ldots, \tilde{\pi}_n)$ is an ϵ-Nash equilibrium.

5.2 High-Level Procedure

Our approach is to compute the best set of punishment strategies τ^* w.r.t. π and check if $\pi \bowtie \tau^*$ is an ϵ-Nash equilibrium. The best punishment strategy against agent j is the one that minimizes its incentive to deviate. To be precise, we define the best response of j with respect to a joint policy $\pi' = (\pi'_1, \ldots, \pi'_n)$ to be $\mathrm{br}_j(\pi') \in \arg\max_{\pi''_j} J_j(\pi'_{-j}, \pi''_j)$. Then, the best set of punishment strategies τ^* w.r.t. π is one that minimizes the value of $\mathrm{br}_j(\pi \bowtie \tau)$ for all $j \in [n]$. To be precise, define $\tau[j] = \{\tau_{ij} \mid i \neq j\}$ to be the set of punishment strategies *against* agent j. Then, we want to compute τ^* such that for all j,

$$\tau^* \in \arg\min_{\tau} J_j((\pi \bowtie \tau)_{-j}, \mathrm{br}_j(\pi \bowtie \tau)). \tag{1}$$

We observe that for any two sets of punishment strategies τ, τ' with $\tau[j] = \tau'[j]$ and any policy π'_j, we have $J_j((\pi \bowtie \tau)_{-j}, \pi'_j) = J_j((\pi \bowtie \tau')_{-j}, \pi'_j)$. This is because, for any τ, punishment strategies in $\tau \backslash \tau[j]$ do not affect the behaviour of the joint policy $((\pi \bowtie \tau)_{-j}, \pi'_j)$, since no agent other than agent j will deviate from π. Hence, $\mathrm{br}_j(\pi \bowtie \tau)$ as well as $J_j((\pi \bowtie \tau)_{-j}, \mathrm{br}_j(\pi \bowtie \tau))$ are independent

of $\tau \setminus \tau[j]$; therefore, we can separately compute $\tau^*[j]$ (satisfying Eq. 1) for each j and take $\tau^* = \bigcup_j \tau^*[j]$. The following theorem follows from the definition of τ^*.

Theorem 3. *Given a finite-state deterministic joint policy $\pi = (\pi_1, \ldots, \pi_n)$, if there is a set of punishment strategies τ such that $\pi \bowtie \tau$ is an ϵ-Nash equilibrium, then $\pi \bowtie \tau^*$ is an ϵ-Nash equilibrium, where τ^* is the set of best punishment strategies w.r.t. π. Furthermore, $\pi \bowtie \tau^*$ is an ϵ-Nash equilibrium iff for all j,*

$$J_j((\pi \bowtie \tau^*)_{-j}, \mathrm{br}_j(\pi \bowtie \tau^*)) - \epsilon \leq J_j(\pi \bowtie \tau^*) = J_j(\pi).$$

Thus, to solve the verification problem, it suffices to compute (or estimate), for all j, the optimal deviation scores

$$\mathtt{dev}_j^{\pi} = \min_{\tau[j]} \max_{\pi'_j} J_j((\pi \bowtie \tau)_{-j}, \pi'_j). \tag{2}$$

5.3 Reduction to Min-Max Games

Next, we describe how to reduce the computation of optimal deviation scores to a standard self-play RL setting. We first translate the problem from the specification setting to a reward-based setting using *reward machines*.

Reward Machines. A *reward machine (RM)* [14] is a tuple $\mathcal{R} = (Q, \delta_u, \delta_r, q_0)$ where Q is a finite set of states, $\delta_u : \mathcal{S} \times \mathcal{A} \times Q \to Q$ is the state transition function, $\delta_r : \mathcal{S} \times Q \to [-1, 1]$ is the reward function and q_0 is the initial RM state. Given a trajectory $\zeta = s_0 \xrightarrow{a_0} \ldots \xrightarrow{a_{t-1}} s_t$, the reward assigned by $\mathcal{R}$ to ζ is $\mathcal{R}(\zeta) = \sum_{k=0}^{t-1} \delta_r(s_k, q_k)$, where $q_{k+1} = \delta_u(s_k, a_k, q_k)$ for all k. For any SPECTRL specification ϕ, we can construct an RM such that the reward assigned to a trajectory ζ indicates whether ζ satisfies ϕ.

Theorem 4. *Given any* SPECTRL *specification ϕ, we can construct an RM $\mathcal{R}_\phi$ such that for any trajectory ζ of length $t+1$, $\mathcal{R}_\phi(\zeta) = \mathbf{1}(\zeta_{0:t} \models \phi)$.*

For an agent j, let $\mathcal{R}_j$ denote $\mathcal{R}_{\phi_j} = (Q_j, \delta_u^j, \delta_r^j, q_0^j)$. Letting $\tilde{\mathcal{D}}_\pi$ be the distribution over length $H+1$ trajectories induced by using π, we have $\mathbb{E}_{\zeta \sim \tilde{\mathcal{D}}_\pi}[\mathcal{R}_j(\zeta)] = J_j(\pi)$. The deviation values defined in Eq. 2 are now min-max values of expected reward, except that it is not in a standard min-max setting since the policy of every non-deviating agent $i \neq j$ is constrained to be of the form $\pi_i \bowtie \tau_i$. This issue can be handled by considering a product of $\mathcal{M}$ with the reward machine $\mathcal{R}_j$ and the finite-state joint policy π. The following theorem follows naturally.

Theorem 5. *Given a finite-state deterministic joint policy $\pi = (M, \alpha, \sigma, m_0)$, for any agent j, we can construct a simulator for an augmented two-player zero-sum Markov game $\mathcal{M}_j^{\pi}$ (with rewards) which has the following properties.*

- *The number of states in $\mathcal{M}_j^{\pi}$ is at most $2|S||M||Q_j|$.*
- *The actions of player 1 is A_j, and the actions of player 2 is $\mathcal{A}_{-j} = \prod_{i \neq j} A_i$.*

Algorithm 2 VERIFYNASH

Inputs: Finite-state deterministic joint policy π, specifications ϕ_j for all j, Nash factor ϵ, precision δ, failure probability p.

Outputs: `True` or `False` along with a set of punishment strategies τ.

`existsNE` $\leftarrow$ `True`
$\tau \leftarrow \emptyset$
$\tilde{\mathcal{M}} \leftarrow$ BFS-ESTIMATE$(\mathcal{M}, \delta, p)$ // Only run if $\mathcal{M}$ has not been estimated before.
for agent $j \in \{1, \ldots, n\}$ **do**
  $\mathcal{R}_j \leftarrow$ CONSTRUCTRM(ϕ_j)
  $\tilde{\mathcal{M}}_j \leftarrow$ CONSTRUCTGAME$(\tilde{\mathcal{M}}, j, \mathcal{R}_j, \pi)$
  $\tilde{\texttt{dev}}_j \leftarrow \min_{\bar{\pi}_2} \max_{\bar{\pi}_1} \bar{J}^{\tilde{\mathcal{M}}_j}(\bar{\pi}_1, \bar{\pi}_2)$
  $\bar{\pi}_2^* \leftarrow \arg\min_{\bar{\pi}_2} \max_{\bar{\pi}_1} \bar{J}^{\tilde{\mathcal{M}}_j}(\bar{\pi}_1, \bar{\pi}_2)$
  `existsNE` $\leftarrow$ `existsNE` $\wedge$ $(\tilde{\texttt{dev}}_j \leq J_j(\pi) + \epsilon - \delta)$
  $\tau \leftarrow \tau \cup$ PUNSTRAT$(\bar{\pi}_2^*)$
return `existsNE`, τ

- *The min-max value of the two player game corresponds to the deviation cost of j, i.e.,*

$$\texttt{dev}_j^{\pi} = \min_{\bar{\pi}_2} \max_{\bar{\pi}_1} \bar{J}_j^{\pi}(\bar{\pi}_1, \bar{\pi}_2),$$

  where $\bar{J}_j^{\pi}(\bar{\pi}_1, \bar{\pi}_2) = \mathbb{E}\big[\sum_{k=0}^{H} R_j(\bar{s}_k, a_k) \mid \bar{\pi}_1, \bar{\pi}_2\big]$ is the expected sum of rewards w.r.t. the distribution over $(H+1)$-length trajectories generated by using the joint policy $(\bar{\pi}_1, \bar{\pi}_2)$ in $\mathcal{M}_j^{\pi}$.
- *Given any policy $\bar{\pi}_2$ for player 2 in $\mathcal{M}_j^{\pi}$, we can construct a set of punishment strategies $\tau[j] =$ PUNSTRAT$(\bar{\pi}_2)$ against agent j in $\mathcal{M}$ such that*

$$\max_{\bar{\pi}_1} \bar{J}_j^{\pi}(\bar{\pi}_1, \bar{\pi}_2) = \max_{\pi_j'} J_j((\pi \bowtie \tau[j])_{-j}, \pi_j').$$

Given an estimate $\tilde{\mathcal{M}}$ of $\mathcal{M}$, we can also construct an estimate $\tilde{\mathcal{M}}_j^{\pi}$ of $\mathcal{M}_j^{\pi}$.

We omit the superscript π from $\mathcal{M}_j^{\pi}$ when there is no ambiguity. We denote by CONSTRUCTGAME$(\tilde{\mathcal{M}}, j, \mathcal{R}_j, \pi)$ the product construction procedure that constructs and returns $\tilde{\mathcal{M}}_j^{\pi}$.

5.4 Solving Min-Max Games

The min-max game $\mathcal{M}_j$ can be solved using self-play RL algorithms. Many of these algorithms provide probabilistic approximation guarantees for computing the min-max value of the game. We use a model-based algorithm, similar to the one proposed in [4], that first estimates the model $\mathcal{M}_j$ and then solves the game in the estimated model.

One approach is to use existing algorithms for reward-free exploration to estimate the model [15], but this approach requires estimating each $\mathcal{M}_j$ separately. Under Assumption 1, we provide a simpler and more sample-efficient algorithm, called BFS-ESTIMATE, for estimating $\mathcal{M}$. BFS-ESTIMATE performs a search

over the transition graph of $\mathcal{M}$ by exploring previously seen states in a breadth first manner. When exploring a state s, multiple samples are collected by taking all possible actions in s several times and the corresponding transition probabilities are estimated. After obtaining an estimate of $\mathcal{M}$, we can directly construct an estimate of $\mathcal{M}_j^\pi$ for any π and j when required. Letting $|Q| = \max_j |Q_j|$ and $|M|$ denote the size of the largest finite-state policy output by our enumeration algorithm, we get the following guarantee.

Theorem 6. *For any $\delta > 0$ and $p \in (0, 1]$,* BFS-ESTIMATE$(\mathcal{M}, \delta, p)$ *computes an estimate $\tilde{\mathcal{M}}$ of $\mathcal{M}$ using $O\left(\frac{|\mathcal{S}|^3|M|^2|Q|^4|\mathcal{A}|H^4}{\delta^2} \log\left(\frac{|\mathcal{S}||\mathcal{A}|}{p}\right)\right)$ sample steps such that with probability at least $1-p$, for any finite-state deterministic joint policy π and any agent j,*

$$\left| \min_{\bar{\pi}_2} \max_{\bar{\pi}_1} \bar{J}^{\tilde{\mathcal{M}}_j^\pi}(\bar{\pi}_1, \bar{\pi}_2) - \boldsymbol{dev}_j^\pi \right| \leq \delta,$$

where $\bar{J}^{\tilde{\mathcal{M}}_j^\pi}(\bar{\pi}_1, \bar{\pi}_2)$ is the expected reward over length $H+1$ trajectories generated by $(\bar{\pi}_1, \bar{\pi}_2)$ in $\tilde{\mathcal{M}}_j^\pi$. Furthermore, letting $\bar{\pi}_2^* \in \arg\min_{\bar{\pi}_2} \max_{\bar{\pi}_1} \bar{J}^{\tilde{\mathcal{M}}_j}(\bar{\pi}_1, \bar{\pi}_2)$ and $\tau[j] =$ PUNSTRAT$(\bar{\pi}_2^*)$*, we have*

$$\left| \max_{\bar{\pi}_1} \bar{J}^{\tilde{\mathcal{M}}_j^\pi}(\bar{\pi}_1, \bar{\pi}_2^*) - \max_{\pi_j'} J_j((\pi \bowtie \tau[j])_{-j}, \pi_j') \right| \leq \delta. \tag{3}$$

The min-max value of $\tilde{\mathcal{M}}_j^\pi$ as well as $\bar{\pi}_2^*$ can be computed using value iteration. Our full verification algorithm is summarized in Algorithm 2. It checks if $\tilde{\texttt{dev}}_j \leq J_j(\pi) + \epsilon - \delta$ for all j, and returns `True` if so and `False` otherwise. It also simultaneously computes the punishment strategies τ using the optimal policies for player 2 in the punishment games. Note that BFS-ESTIMATE is called only once (i.e., the first time VERIFYNASH is called) and the obtained estimate $\tilde{\mathcal{M}}$ is stored and used for verification of every candidate policy π. The following soundness guarantee follows from Theorem 6.

Corollary 1 (Soundness). *For any $p \in (0, 1]$, $\varepsilon > 0$ and $\delta \in (0, \varepsilon)$, with probability at least $1-p$, if* HIGHNASHSEARCH *returns a joint policy $\tilde{\pi}$ then $\tilde{\pi}$ is an ϵ-Nash equilibrium.*

6 Complexity

In this section, we analyze the time and sample complexity of our algorithm in terms of the number of agents n, size of the specification $|\phi| = \max_{i \in [n]} |\phi_i|$, number of states in the environment $|\mathcal{S}|$, number of joint actions $|\mathcal{A}|$, time horizon H, precision δ and the failure probability p.

Sample Complexity. It is known [16] that the number of edges in the abstract graph $\mathcal{G}_i$ corresponding to specification ϕ_i is $O(|\phi_i|^2)$. Hence for any set of active agents B, the number of edges in the product abstract graph $\mathcal{G}_B$ is $O(|\phi|^{2|B|})$.

Hence total number of edge policies learned by our compositional RL algorithm is $\sum_{B\subseteq[n]} O((|\phi|^2)^{|B|}) = O((|\phi|^2+1)^n)$. We learn each edge using a fixed number of sample steps C, which is a hyperparameter.

The number of samples used in the verification phase is the same as the number used by BFS-Estimate. The maximum size of a candidate policy output by the enumeration algorithm $|M|$ is at most the length of the longest path in a product abstract graph. Since the maximum path length in a single abstract graph $\mathcal{G}_i$ is bounded by $|\phi_i|$ and at least one agent must progress along every edge in a product graph, the maximum length of a path in any product graph is at most $n|\phi|$. Also, the number of states in the reward machine $\mathcal{R}_j$ corresponding to $|\phi_j|$ is $O(2^{|\phi_j|})$. Hence, from Theorem 6 we get that the total number of sample steps used by our algorithm is $O\big((|\phi|^2+1)^n C + \frac{2^{4|\phi|}|\mathcal{S}|^3 n^2 |\phi|^2 |\mathcal{A}| H^4}{\delta} \log\big(\frac{|\mathcal{S}||\mathcal{A}|}{p}\big)\big)$.

Time Complexity. As with sample complexity, the time required to learn all edge policies is $O((|\phi|^2+1)^n(C+|\mathcal{A}|))$ where the term $|\mathcal{A}|$ is added to account for the time taken to select an action from $\mathcal{A}$ during exploration (we use Q-learning with ε-greedy exploration for learning edge policies). Similarly, time taken for constructing the reward machines and running BFS-Estimate is $O(\frac{2^{4|\phi|}|\mathcal{S}|^3 n^2 |\phi|^2 |\mathcal{A}| H^4}{\delta} \log\big(\frac{|\mathcal{S}||\mathcal{A}|}{p}\big))$.

The total number of path policies considered for a given set of active agents B is bounded by the number of paths in the product abstract graph $\mathcal{G}_B$ that terminate in a final product state. First, let us consider paths in which exactly one agent progresses in each edge. The number of such paths is bounded by $(|B||\phi|)^{|B||\phi|}$ since the length of such paths is bounded by $|B||\phi|$ and there are at most $|B||\phi|$ choices at each step—i.e., progressing agent j and next vertex of the abstract graph $\mathcal{G}_{\phi_j}$. Now, any path in $\mathcal{G}_B$ can be constructed by merging adjacent edges along such a path (in which at most one agent progresses at any step). The number of ways to merge edges along such a path is bounded by the number of groupings of edges along the path into at most $|B||\phi|$ groups which is bounded by $(|B||\phi|)^{|B||\phi|}$. Therefore, the total number of paths in $\mathcal{G}_B$ is at most $2^{2|B||\phi|\log(n|\phi|)}$. Finally, the total number of path policies considered is at most $\sum_{B\subseteq[n]} 2^{2|B||\phi|\log(n|\phi|)} \leq ((n|\phi|)^{2|\phi|}+1)^n = O(2^{2n|\phi|\log(2n|\phi|)})$.

Now, for each path policy π, the verification algorithm solves $\tilde{\mathcal{M}}_j^{\pi}$ using value iteration which takes $O(|\tilde{\mathcal{S}}||\mathcal{A}|Hf(|\mathcal{A}|)) = O(2^{|\phi|}n|\phi||\mathcal{S}||\mathcal{A}|Hf(|\mathcal{A}|))$ time, where $f(|\mathcal{A}|)$ is the time required to solve a linear program of size $|\mathcal{A}|$. Also accounting for the time taken to sort the path policies, we arrive at a time complexity bound of $2^{O(n|\phi|\log(n|\phi|))}\text{poly}(|\mathcal{S}|, |\mathcal{A}|, H, \frac{1}{p}, \frac{1}{\delta})$.

It is worth noting that the procedure halts as soon as our verification procedure successfully verifies a policy; this leads to early termination for cases where there is a high value ϵ-Nash equilibrium (among the policies considered). Furthermore, our verification algorithm runs in polynomial time and therefore one could potentially improve the overall time complexity by reducing the search space in the prioritized enumeration phase—e.g., by using domain specific insights.

7 Experiments

We evaluate our algorithm on finite state environments and a variety of specifications, aiming to answer the following:

- Can our approach be used to learn ϵ-Nash equilibria?
- Can our approach learn policies with high social welfare?

We compare our approach to two baselines described below, using two metrics: (i) the social welfare $\texttt{welfare}(\pi)$ of the learned joint policy π, and (ii) an estimate of the minimum value of ϵ for which π forms an ϵ-Nash equilibrium:

$$\epsilon_{\min}(\pi) = \max\{J_i(\pi_{-i}, \mathrm{br}_i(\pi)) - J_i(\pi) \mid i \in [n]\}.$$

Here, $\epsilon_{\min}(\pi)$ is computed using single agent RL (specifically, Q-learning) to compute $\mathrm{br}_i(\pi)$ for each agent i.

Environments and Specifications. We show results on the *Intersection environment* illustrated in Fig. 1, which consists of k-cars (agents) at a 2-way intersection of which k_1 and k_2 cars are placed along the N-S and E-W axes, respectively. The state consists of the location of all cars where the location of a single car is a non-negative integer. 1 corresponds to the intersection, 0 corresponds to the location one step towards the south or west of the intersection (depending on the car) and locations greater than 1 are to the east or north of the intersection. Each agent has two actions. `STAY` stays at the current position. `MOVE` decreases the position value by 1 with probability 0.95 and stays with probability 0.05. We consider specifications similar to the ones in the motivating example.

Baselines. We compare our NE computation method (HighNashSearch) to two approaches for learning in non-cooperative games. The first, maqrm, is an adaption of the reward machine based learning algorithm proposed in [22]. maqrm was originally proposed for cooperative multi-agent RL where there is a single specification for all the agents. It proceeds by first decomposing the specification into individual ones for all the agents and then runs a Q-learning-style algorithm (qrm) in parallel for all the agents. We use the second part of their algorithm directly since we are given a separate specification for each agent. The second baseline, nvi, is a model-based approach that first estimates transition probabilities, and then computes a Nash equilibrium in the estimated game using value iteration for stochastic games [17]. To promote high social welfare, we select the highest value Nash solution for the matrix game at each stage of value iteration. Note that this greedy strategy may not maximize social welfare. Both maqrm and nvi learn from rewards as opposed to specification; thus, we supply rewards in the form of reward machines constructed from the specifications. nvi is guaranteed to return an ϵ-Nash equilibrium with high probability, but maqrm is not guaranteed to do so.

Table 1. Results for all specifications in Intersection Environment. Total of 10 runs per benchmark. Timeout = 24 h.

Spec.	Num. of agents	Algorithm	`welfare`(π) (avg ± std)	$\epsilon_{\min}(\pi)$ (avg ± std)	Num. of terminated runs	Avg. num. of sample steps (in millions)
ϕ^1	3	HighNashSearch	**0.33 ± 0.00**	**0.00 ± 0.00**	10	1.78
		NVI	0.32 ± 0.00	**0.00 ± 0.00**	10	1.92
		MAQRM	0.18 ± 0.01	0.51 ± 0.01	10	2.00
ϕ^2	4	HighNashSearch	**0.55 ± 0.10**	**0.01 ± 0.02**	10	11.53
		NVI	0.04 ± 0.01	0.02 ± 0.01	10	12.60
		MAQRM	0.12 ± 0.01	0.20 ± 0.03	10	15.00
ϕ^3	4	HighNashSearch	**0.49 ± 0.01**	**0.00 ± 0.01**	10	11.26
		NVI	0.45 ± 0.01	**0.00 ± 0.01**	10	12.60
		MAQRM	0.11 ± 0.01	0.22 ± 0.02	10	15.00
ϕ^4	3	HighNashSearch	0.90 ± 0.15	**0.00 ± 0.00**	10	2.16
		NVI	**0.98 ± 0.00**	**0.00 ± 0.00**	4	2.18
		MAQRM	0.23 ± 0.01	0.39 ± 0.04	10	2.00
ϕ^5	5	HighNashSearch	**0.58 ± 0.02**	**0.00 ± 0.00**	10	62.17
		NVI	0.05 ± 0.01	0.01 ± 0.01	7	80.64
		MAQRM	Timeout	Timeout	0	Timeout

Results. Our results are summarized in Table 1. For each specification, we ran all algorithms 10 times with a timeout of 24 h. Along with the average social welfare and $\epsilon_{\min}$, we also report the average number of sample steps taken in the environment as well as the number of runs that terminated before timeout. For a fair comparison, all approaches were given a similar number of samples from the environment.

Nash Equilibrium. Our approach learns policies that have low values of $\epsilon_{\min}$, indicating that it can be used to learn ϵ-Nash equilibria for small values of ϵ. NVI also has similar values of ϵ, which is expected since NVI provides guarantees similar to our approach w.r.t. Nash equilibria computation. On the other hand, MAQRM learns policies with large values of $\epsilon_{\min}$, implying that it fails to converge to a Nash equilibrium in most cases.

Social Welfare. Our experiments show that our approach consistently learns policies with high social welfare compared to the baselines. For instance, ϕ^3 corresponds to the specifications in the motivating example for which our approach learns a joint policy that causes both blue and black cars to achieve their goals. Although NVI succeeds in learning policies with high social welfare for some specifications (ϕ^1, ϕ^3, ϕ^4), it fails to do so for others (ϕ^2, ϕ^5). Additional experiments (see extended version [3]) indicate that NVI achieves similar social

welfare as our approach for specifications in which all agents can successfully achieve their goals (cooperative scenarios). However, in many other scenarios in which only some of the agents can fulfill their objectives, our approach achieves higher social welfare.

8 Conclusions

We have proposed a framework for maximizing social welfare under the constraint that the joint policy should form an ϵ-Nash equilibrium. Our approach involves learning and enumerating a small set of finite-state deterministic policies in decreasing order of social welfare and then using a self-play RL algorithm to check if they can be extended with punishment strategies to form an ϵ-Nash equilibrium. Our experiments demonstrate that our approach is effective in learning Nash equilibria with high social welfare.

One limitation of our approach is that our algorithm does not have any guarantee regarding optimality with respect to social welfare. The policies considered by our algorithm are chosen heuristically based on the specifications, which may lead to scenarios where we miss high welfare solutions. For example, ϕ^2 corresponds to specifications in the motivating example except that the blue car is not required to stay a car length ahead of the other two cars. In this scenario, it is possible for three cars to achieve their goals in an equilibrium solution if the blue car helps the cars behind by staying in the middle of the intersection until they catch up. Such a joint policy is not among the set of policies considered; therefore, our approach learns a solution in which only two cars achieve their goals. We believe that such limitations can be overcome in future work by modifying the various components within our enumerate-and-verify framework.

Acknowledgements. We thank the anonymous reviewers for their helpful comments. This work is supported in part by NSF grant 2030859 to the CRA for the CIFellows Project, ONR award N00014-20-1-2115, DARPA Assured Autonomy award, NSF award CCF 1723567 and ARO award W911NF-20-1-0080.

References

1. Akchurina, N.: Multi-agent reinforcement learning algorithm with variable optimistic-pessimistic criterion. In: ECAI, vol. 178, pp. 433–437 (2008)
2. Alur, R., Bansal, S., Bastani, O., Jothimurugan, K.: A framework for transforming specifications in reinforcement learning. arXiv preprint arXiv:2111.00272 (2021)
3. Alur, R., Bansal, S., Bastani, O., Jothimurugan, K.: Specification-guided learning of Nash equilibria with high social welfare (2022). https://arxiv.org/abs/2206.03348
4. Bai, Y., Jin, C.: Provable self-play algorithms for competitive reinforcement learning. In: Proceedings of the 37th International Conference on Machine Learning (2020)

5. Bouyer, P., Brenguier, R., Markey, N.: Nash equilibria for reachability objectives in multi-player timed games. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 192–206. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15375-4_14
6. Chatterjee, K.: Two-player nonzero-sum ω-regular games. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 413–427. Springer, Heidelberg (2005). https://doi.org/10.1007/11539452_32
7. Chatterjee, K., Majumdar, R., Jurdziński, M.: On nash equilibria in stochastic games. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 26–40. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30124-0_6
8. Czumaj, A., Fasoulakis, M., Jurdzinski, M.: Approximate nash equilibria with near optimal social welfare. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
9. Greenwald, A., Hall, K., Serrano, R.: Correlated Q-learning. In: ICML, Vol.3, pp. 242–249 (2003)
10. Hammond, L., Abate, A., Gutierrez, J., Wooldridge, M.: Multi-agent reinforcement learning with temporal logic specifications. In: International Conference on Autonomous Agents and MultiAgent Systems, pp. 583–592 (2021)
11. Hazan, E., Krauthgamer, R.: How hard is it to approximate the best nash equilibrium? In: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, pp. 720–727. Society for Industrial and Applied Mathematics (2009)
12. Hu, J., Wellman, M.P.: Nash Q-learning for general-sum stochastic games. J. Mach. Learn. Res. **4**(Nov), 1039–1069 (2003)
13. Hu, J., Wellman, M.P., et al.: Multiagent reinforcement learning: theoretical framework and an algorithm. In: ICML, vol. 98, pp. 242–250. Citeseer (1998)
14. Icarte, R.T., Klassen, T., Valenzano, R., McIlraith, S.: Using reward machines for high-level task specification and decomposition in reinforcement learning. In: International Conference on Machine Learning, pp. 2107–2116. PMLR (2018)
15. Jin, C., Krishnamurthy, A., Simchowitz, M., Yu, T.: Reward-free exploration for reinforcement learning. In: International Conference on Machine Learning, pp. 4870–4879. PMLR (2020)
16. Jothimurugan, K., Bansal, S., Bastani, O., Alur, R.: Compositional reinforcement learning from logical specifications. Adv. Neural Inf. Proc. Syst. **34**, 10026–10039 (2021)
17. Kearns, M., Mansour, Y., Singh, S.: Fast planning in stochastic games. In: Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence, pp. 309–316 (2000)
18. Kwiatkowska, M., Norman, G., Parker, D., Santos, G.: Equilibria-based probabilistic model checking for concurrent stochastic games. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) FM 2019. LNCS, vol. 11800, pp. 298–315. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30942-8_19
19. Kwiatkowska, M., Norman, G., Parker, D., Santos, G.: PRISM-games 3.0: stochastic game verification with concurrency, equilibria and time. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12225, pp. 475–487. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53291-8_25
20. Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: Machine Learning Proceedings 1994, pp. 157–163. Elsevier (1994)
21. Littman, M.L.: Friend-or-foe Q-learning in general-sum games. In: ICML, vol. 1, pp. 322–328 (2001)

22. Neary, C., Xu, Z., Wu, B., Topcu, U.: Reward machines for cooperative multi-agent reinforcement learning (2021)
23. Perolat, J., Strub, F., Piot, B., Pietquin, O.: Learning nash equilibrium for general-sum Markov games from batch data. In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (2017)
24. Prasad, H., LA, P., Bhatnagar, S.: Two-timescale algorithms for learning nash equilibria in general-sum stochastic games. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, pp. 1371–1379 (2015)
25. Shapley, L.S.: Stochastic games. Proc. Nat. Acad. Sci. **39**(10), 1095–1100 (1953)
26. Wei, C.Y., Hong, Y.T., Lu, C.J.: Online reinforcement learning in stochastic games. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 4994–5004 (2017)
27. Zinkevich, M., Greenwald, A., Littman, M.: Cyclic equilibria in markov games. Adv. Neural Inf. Proc. Syst. **18**, 1641 (2006)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.