Concurrent Multiagent Reinforcement Learning with Reward Machines

Alessandro Trapasso[a,*] **and Anders Jonsson**[b]

[a]Sapienza University of Rome, Rome, Italy
[b]Universitat Pompeu Fabra, Barcelona, Spain

Abstract. Coordinating and synchronizing multiple agents in reinforcement learning (RL) presents significant challenges, particularly when concurrent actions and shared objectives are required. We propose a novel framework that integrates Reward Machines (RMs) with Partial-Order Planning (POP) to enhance coordination in multiagent reinforcement learning (MARL). By transforming high-level POP strategies into individual RMs for each agent, our approach explicitly captures action dependencies and concurrency requirements, enabling agents to learn and execute coordinated plans effectively in complex environments. We validate our approach in a grid-based multiagent domain in which agents have to synchronize actions such as jointly accessing limited pathways or collaboratively manipulating objects. The explicit representation of action dependencies and synchronization points in RMs provides a scalable and flexible mechanism to model concurrent actions, enabling agents to focus on relevant tasks and reducing exploration.

1 Introduction

In recent years, reinforcement learning (RL) has emerged as the method of choice for solving complex sequential decision problems. RL algorithms have accomplished a number of groundbreaking results, achieving superhuman performance in complex board games such as Go, chess and shogi [22]. Successful real-world applications include energy-saving decision strategies for data center cooling [13], navigating stratospheric balloons [2], controlling the tokamak plasma in fusion reactors [5], and incorporating human feedback when training large language models [18].

In cooperative multiagent RL (MARL) [1], multiple agents collaborate in order to solve a sequential decision problem. Each agent typically has its own sensors and actuators, but to achieve near-optimal behavior the agents have to coordinate their actions carefully to maximize a common reward signal. Several paradigms have been proposed for solving cooperative MARL. In this paper we focus on the paradigm called centralized training for decentralized execution, in the fully observable setting. This implies that during training, a single decision maker can observe the complete environment state and control the actions of all agents. During execution, however, each agent selects actions independently with little input from other agents.

Centralized training is challenging due to the combinatorial aspect of MARL: the number of states and actions is exponential in the number of agents of the problem. For this reason, state-of-the-art MARL algorithms typically exploit the problem structure in an attempt to subdivide the set of agents into smaller teams, each of which learn their own partial decisions. Global coordination is then achieved by communicating or coordinating the decisions of the different agent teams. Such structure implies that the MARL problem is loosely coupled, or equivalently, that the agent coordination graph is sparse [10]. Loose coupling often takes the form of independent dynamics with shared reward, or solving a set of tasks that can be distributed among the agents.

In this work we tackle the challenging setting in which the MARL problem is strongly coupled, in the sense that the effect of a joint action depends critically on all agents involved. In many real-world applications, agents have to jointly manipulate objects, e.g., lift, push, or maneuver various items together. Hence a single agent cannot perform such actions on its own. In this setting, any agent may need to cooperate with any other agent in order to solve the problem, implying that the agent coordination graph is dense. Since this makes it difficult to form smaller teams of agents and solve the problem in a distributed manner, state-of-the-art MARL algorithms do not scale well to such problems as the number of agents grows.

To successfully handle strongly coupled MARL, our approach is to assume that a MARL problem has an associated high-level problem description in the form of a multiagent planning problem (MAP). Concretely, the MAP efficiently encodes all relevant joint actions of agents, but does not model individual actions that agents can perform on their own. Such a MAP also suffers from an exponential number of actions, but researchers have proposed reductions to single-agent planning problems that can be solved efficiently by state-of-the-art planners, thereby scaling to much larger teams of agents.

1.1 Contributions

In this work we propose a novel approach for solving MARL problems, similar to the paradigm of centralized training for decentralized execution. However, our approach does not perform reinforcement learning jointly for all agents, and the centralized training step is replaced by a planning algorithm that solves the high-level MAP. Concretely, we use the concept of affordances to model joint actions and translate the MAP to a single-agent problem [4]. We then apply the FMAP planner [29] to produce a partial-order plan (POP) that solves the single-agent problem. A POP is a directed graph that includes temporal dependencies between actions [21, 32]. From the POP we construct a set of reward machines (RMs), one for each agent. Given the reward machines and the individual dynamics of each agent, we can now apply reinforcement learning in a distributed manner, each agent learning to complete its part of the plan. During execution, each

* Corresponding author. Email: trapasso@diag.uniroma1.it.

agent selects actions on their own, and joint actions are automatically triggered when their preconditions are satisfied.

The novel contributions of our work include the following:

- A method for transforming POPs into RMs, providing a formal execution model of concurrent tasks and action dependencies crucial for efficient distributed reinforcement learning and execution.
- A flexible framework supporting agents with private and public actions, facilitating decentralized learning and coordination in cooperative MARL. This framework offers a practical approach to managing complex, concurrent actions in multiagent environments, improving scalability and applicability to real-world scenarios where collaboration and concurrency are essential.

Our approach is based on three key assumptions outlined below:

1. Each MARL problem that we wish to solve has an associated multiagent planning problem (MAP) description that includes a goal condition and a model of all high-level public actions. Having access to a MAP description makes it possible to solve the problem using state-of-the-art goal-directed MAP planners, obviating the need for random exploration which can be very costly when considering exponentially large joint action spaces.
2. Each public action has deterministic effects, though the private actions of each agent can have stochastic effects. The reason for having deterministic public actions is that deterministic planners are considerably more powerful than non-deterministic planners, which impacts the scalability of MAP planning.
3. Each agent has a no-op action that preserves the current state of the agent, i.e., there are no external effects on the agent's state. This assumption is needed for our execution model to work, since an agent often has to wait for other agents to finish their current subtask before the agents can take a joint action together.

We believe that the second and third assumption can be relaxed in future work, but the first assumption is the backbone of our approach to MARL. Due to the computational complexity, we compute a satisficing solution to the MAP problem rather than an optimal solution.

1.2 Related Work

Almost all previous work in cooperative multiagent planning assumes that agents can perform actions sequentially, e.g. the Competition of Distributed and Multiagent Planners [25] and FMAP [29]. Hence concurrency is not needed. A few researchers have developed algorithms for solving concurrent multiagent planning problems [4, 17, 15, 7, 23], one of which we exploit in this work. However, these algorithms cannot handle stochastic actions present in RL.

In the single-agent setting, there exists previous work that solves a high-level planning problem to guide RL [11], and similar ideas have been proposed for loosely coupled MARL [24]. As for RMs, researchers have proposed extensions to MARL in the case for which the RM of each agent is hand-crafted [16]. There also exists previous work that synthesizes RMs from a logic specification of the high-level problem in the loosely coupled setting [30]. We believe our work is the first to apply this idea to strongly coupled cooperative MARL. In addition, handling concurrency requires a refined execution model of RMs that automatically applies joint actions when they become applicable.

The state-of-the-art in cooperative MARL is to distribute learning among agents, e.g. value function factorization [26, 20, 31] and centralized critic actor-critic [14, 6, 33]. These works assume that each agent has its own policy and/or value function, and include a central collaboration mechanism to ensure that the resulting policy is coordinated. However, distributing learning is insufficient to solve the type of strongly coupled MARL problems that we consider in this paper.

2 Background

In this section we introduce concepts and notation used throughout the paper. Let $\mathbb{N}$ be the set of natural numbers excluding 0. Given $n \in \mathbb{N}$, we use $[\![n]\!]$ to denote the set $\{1, \dots, n\}$. Given a set $\mathcal{X}$, we use $\Delta(\mathcal{X}) = \{p \in \mathbb{R}^{\mathcal{X}} : \sum_{x \in \mathcal{X}} p_x = 1, p_x \geq 0 \; \forall x \in \mathcal{X}\}$ to denote the probability simplex, i.e. the set of probability distributions on $\mathcal{X}$.

2.1 Multiagent Reinforcement Learning

We define a multiagent Markov decision process (MMDP) as a tuple $\mathcal{M} = \langle N, S, \{A_n\}_{n \in [\![N]\!]}, P, R \rangle$, where N is the number of agents, S is the shared state space, A_n is the action set of agent $n \in [\![N]\!]$, $P : S \times A \to \Delta(S)$ is a transition kernel, and $R : S \times A \to \mathbb{R}$ is a reward function. Both P and R are defined on the joint action space $A = A_1 \times \cdots \times A_N$. The aim is to learn a joint policy $\pi : S \to \Delta(A)$, a mapping from states to probability distributions on joint actions, that maximizes expected future reward. We assume that the problem that we wish to solve is expressed as an MMDP $\mathcal{M}$.

2.2 Multiagent Planning

We define planning problems in first-order logic, similar to PDDL [9]. A planning domain $\mathcal{D} = \langle \Psi, \mathcal{A} \rangle$ consists of a predicate set Ψ and an action set $\mathcal{A}$. Each predicate and action has an arity. Given an action $a \in \mathcal{A}$ with arity $k \in \mathbb{N}$, let $\mathcal{X} = \{x_1, \dots, x_k\}$ be a set of variable symbols, and let $\mathcal{T}(\Psi, \mathcal{X})$ be the set of first-order terms formed by assigning variable symbols in $\mathcal{X}$ to predicates in Ψ. Action a has a precondition $\text{Pre}(a) \subseteq \mathcal{T}(\Psi, \mathcal{X})$, an add effect $\text{Add}(a) \subseteq \mathcal{T}(\Psi, \mathcal{X})$ and a delete effect $\text{Del}(a) \subseteq \mathcal{T}(\Psi, \mathcal{X})$.

Given a planning domain $\mathcal{D} = \langle \Psi, \mathcal{A} \rangle$, a multiagent planning problem (MAP) is a tuple $\mathcal{P} = \langle N, \mathcal{C}, I, G \rangle$, where N is a number of agents, $\mathcal{C}$ is a set of constant symbols called objects, I is an initial state and G is a goal condition. The object set $\mathcal{C}$ induces a fluent set $F \subseteq \mathcal{T}(\Psi, \mathcal{C})$ and an operator set $O \subseteq \mathcal{T}(\mathcal{A}, \mathcal{C})$, i.e. first-order atoms obtained by assigning objects in $\mathcal{C}$ to predicates in Ψ and actions in $\mathcal{A}$, respectively. A state $s \subseteq F$ is a subset of fluents that are true, while all other fluents are assumed to be false. Specifically, $I \subseteq F$ and $G \subseteq F$ are both subsets of fluents.

We assume that $\mathcal{C}$ contains integers representing the agents, i.e. $[\![N]\!] \subseteq \mathcal{C}$. We restrict each fluent $f = p(c_1, \dots, c_m)$, $p \in \Psi$, and operator $o = a(c_1, \dots, c_k)$, $a \in \mathcal{A}$, such that $c_i \in \mathcal{C} \setminus [\![N]\!]$, $i > 1$. Hence only the first argument c_1 can be an agent, and is required to be for operators. This allows us to partition the fluents and operators as $F = F_1 \cup \cdots \cup F_N \cup F_{pub}$ and $O = O_1 \cup \cdots \cup O_N$, where F_n and O_n, $n \in [\![N]\!]$, are the fluents and operators such that $c_1 = n$, respectively, and F_{pub} is the set of public fluents not associated with any agent. We can give agents different capabilities by carefully defining the predicates and preconditions of actions.

An operator $o = a(c_1, \dots, c_k)$, $a \in \mathcal{A}$, has precondition $\text{Pre}(o) \subseteq F$, add effect $\text{Add}(o) \subseteq F$ and delete effect $\text{Del}(o) \subseteq F$, derived from $\text{Pre}(a)$, $\text{Add}(a)$ and $\text{Del}(a)$ by replacing each variable symbol x_i of a with the associated object c_i. A joint operator $\mathbf{o} = (o_1, \dots, o_N) \in O_1 \times \cdots \times O_N$ consists of one operator per agent. The precondition and effects are defined as $\text{Pre}(\mathbf{o}) = \bigcup_{n \in [\![N]\!]} \text{Pre}(o_n)$, $\text{Add}(\mathbf{o}) = \bigcup_{n \in [\![N]\!]} \text{Add}(o_n)$ and $\text{Del}(\mathbf{o}) = \bigcup_{n \in [\![N]\!]} \text{Del}(o_n)$, and $\mathbf{o}$ is well-defined if $\text{Add}(\mathbf{o}) \cap \text{Del}(\mathbf{o}) = \emptyset$, i.e. if it does not add and delete the same fluent. To model arbitrary interactions, we define a no-op action `noop` with arity 1 and

$\text{Pre}(\texttt{noop}) = \text{Add}(\texttt{noop}) = \text{Del}(\texttt{noop}) = \emptyset$. This allows us to define joint operators $(\texttt{noop}(1), \ldots, a(n, c_2, \ldots, c_k), \ldots, \texttt{noop}(N))$ in which a single agent n is acting (or a subset of agents). Without loss of generality, we often write such a joint operator simply as $a(n, c_2, \ldots, c_k)$, assuming that n acts alone.

A joint operator $\mathbf{o}$ is applicable in state s if and only if $\text{Pre}(\mathbf{o}) \subseteq s$, and applying $\mathbf{o}$ in s results in a new state $s \oplus \mathbf{o} = (s \setminus \text{Del}(\mathbf{o})) \cup \text{Add}(\mathbf{o})$. A plan $\pi = (\mathbf{o}_1, \ldots, \mathbf{o}_\ell)$ is a sequence of joint operators which induces a state sequence $s_0, \ldots, s_\ell$ such that $s_0 = I$ and, for each $i \in [\![\ell]\!]$, $\mathbf{o}_i$ is applicable in s_{i-1} and results in state $s_{i-1} \oplus \mathbf{o}_i = s_i$. A plan π solves $\mathcal{P}$ if and only if $G \subseteq s_\ell$, i.e. if state s_ℓ satisfies the goal condition.

Example 1. *Let $\mathcal{D} = \langle \Psi, \mathcal{A} \rangle$ be a planning domain defined by predicates $\Psi = \{\texttt{at}_2, \texttt{on}_2, \texttt{pen}_1, \texttt{box}_1\}$ and actions $\mathcal{A} = \{\texttt{move}_3, \texttt{pick}_3, \texttt{drop}_3, \texttt{push}_4\}$, where subscripts denote arity. The actions are defined as follows:*

Action	*Pre*	*Add*	*Del*
$\texttt{move}(x, y, z)$	$\{\texttt{at}(x, y)\}$	$\{\texttt{at}(x, z)\}$	$\{\texttt{at}(x, y)\}$
$\texttt{pick}(x, y, z)$	$\{\texttt{at}(x, z), \texttt{at}(y, z), \texttt{pen}(y)\}$	$\{\texttt{on}(x, y)\}$	$\{\texttt{at}(y, z)\}$
$\texttt{drop}(x, y, z)$	$\{\texttt{at}(x, z), \texttt{on}(x, y), \texttt{pen}(y)\}$	$\{\texttt{at}(y, z)\}$	$\{\texttt{on}(x, y)\}$
$\texttt{push}(x, y, z, \ell)$	$\{\texttt{at}(x, z), \texttt{at}(y, z), \texttt{box}(y)\}$	$\{\texttt{at}(x, \ell), \texttt{at}(y, \ell)\}$	$\{\texttt{at}(x, z), \texttt{at}(y, z)\}$

An example MAP is $\mathcal{P} = \langle 2, \{1, 2, a, b, c, q, r\}, I, G \rangle$ with $I = \{\texttt{at}(1, a), \texttt{at}(2, a), \texttt{at}(q, a), \texttt{at}(r, a), \texttt{pen}(q), \texttt{box}(r)\}$ and $G = \{\texttt{at}(q, c), \texttt{at}(r, b)\}$. This MAP has 2 agents, 3 locations a, b and c, a pen q and a box r. Initially the agents and objects are at a, and the goal is to move q to c and r to b. Some examples of operators include $\texttt{move}(1, a, c)$, $\texttt{pick}(1, q, a)$, $\texttt{drop}(1, q, c)$ and $\texttt{push}(1, r, a, b)$.

2.3 Concurrency Constraints

Even if a joint operator $\mathbf{o}$ is well-defined, the problem definition may prevent its application. In Example 1, the joint operator $(\texttt{pick}(1, q, a), \texttt{pick}(2, q, a))$ is well-defined but results in both agents holding the pen. If we assume that the box is heavy, both agents have to push it simultaneously, so the joint operator $(\texttt{push}(1, r, a, b), \texttt{noop}(2))$ is not applicable.

Researchers have proposed different forms of concurrency constraints to model joint operator applicability, e.g. explicitly include operators in preconditions [3, 12], or associate an affordance with each action [4]. We adopt the latter in this work, associating each action $a \in \mathcal{A}$ with an affordance $[l, u]$, where l and u are lower and upper bounds on the number of agents in a joint operator. In Example 1, `move` has affordance $[1, N]$, `pick` and `drop` have affordance $[1, 1]$, while `push` has affordance $[2, N]$. The affordance only applies to operators with the same arguments, i.e. $(\texttt{pick}(1, q1, a), \texttt{pick}(2, q2, a))$ would be applicable. Operators from different actions are always assumed jointly applicable.

Since the number of joint operators is exponential in N, planning is costly. To alleviate the complexity, researchers have proposed reductions from MAPs to single-agent planning [4, 17, 15, 7, 23]. A single-agent problem $\mathcal{P} = \langle 1, \mathcal{C}, I, G \rangle$ is a special case of a MAP. The translation from MAP planning to single-agent planning is sound, in the sense that a single-agent solution can always be translated to a solution of the original MAP [4, 7]. To handle affordances, joint operators are serialized by introducing start and end actions [4]. In Example 1, the joint operator $(\texttt{push}(1, r, a, b), \texttt{push}(2, r, a, b))$ is simulated by applying the operators in sequence between `start-push` and `end-push`, the latter ensuring that the affordance is satisfied. Though serialization does not directly reduce the search space, state-of-the-art planners can effectively navigate huge search spaces when the number of operators is limited.

2.4 Partial-Order Planning

Given a single-agent problem defined by $\mathcal{D} = \langle \Psi, \mathcal{A} \rangle$ and $\mathcal{P} = \langle 1, \mathcal{C}, I, G \rangle$, we introduce dummy operators o_I and o_G defined as $\text{Pre}(o_I) = \text{Del}(o_I) = \text{Add}(o_G) = \text{Del}(o_G) = \emptyset$, $\text{Add}(o_I) = I$, and $\text{Pre}(o_G) = G$, i.e. o_I adds I while o_G has precondition G. Since operators may appear multiple times in a partial-order plan, an operator instance is a pair (o, k) of an operator o and an integer $k \in \mathbb{N}$ identifying the instance.

Given $\mathcal{P}$, a partial-order plan (POP) $\pi_{POP} = \langle \mathcal{G}, L \rangle$ consists of a directed graph $\mathcal{G} = (V, E)$ with operator instances as nodes, and an associated set of causal links L. Concretely, $V = \{(o_I, 1), (o_G, 1)\} \cup V_O$ contains one instance each of o_I and o_G, as well as an arbitrary set of instances of other operators $V_O \subseteq O \times \mathbb{N}$. Each causal link $((o_1, k_1), f, (o_2, k_2)) \in L$ involves two operator instances in V and a fluent $f \in \text{Add}(o_1) \cap \text{Pre}(o_2)$, indicating that the operator instance (o_1, k_1) adds the precondition f of (o_2, k_2). Each causal link $((o_1, k_1), f, (o_2, k_2)) \in L$ induces a corresponding graph edge $((o_1, k_1), (o_2, k_2)) \in E$. In addition, for each operator instance $(o_3, k_3) \in V$ such that $f \in \text{Del}(o_3)$, E contains either $((o_3, k_3), (o_1, k_1))$ or $((o_2, k_2), (o_3, k_3))$, ensuring that f is not deleted between (o_1, k_1) and (o_2, k_2).

For π_{POP} to solve $\mathcal{P}$, $\mathcal{G}$ has to be acyclic with source node $(o_I, 1)$ and sink node $(o_G, 1)$, and L has to contain a causal link $((o_1, k_1), f, (o_2, k_2))$ for each precondition $f \in \text{Pre}(o_2)$ of each operator instance $(o_2, k_2) \in V$, ensuring that all preconditions are satisfied. Specifically, this implies that the precondition G of $(o_G, 1)$ is satisfied, ensuring that the goal condition G of $\mathcal{P}$ holds. Any total ordering of the action instances in V that is consistent with E will solve $\mathcal{P}$ [32], so a topological sort of $\mathcal{G}$ recovers a sequential plan π.

2.5 Reward Machines

We define a simple reward machine (RM) [28] as a tuple $\mathcal{R} = \langle U, \Sigma, u^0, u^A, \delta, \varphi \rangle$, where U is a finite set of RM states, Σ is a finite set of symbols, $u^0 \in U$ is an initial RM state, $u^A \in U$ is an accepting RM state, $\delta : U \times \Sigma \to U$ is a transition function, and $\varphi : U \times U \to \{0, 1\}$ is an output function. We assume that u^A is a sink state, that $\varphi(u, u^A) = 1$ for each $u \in U \setminus \{u^A\}$, and that $\varphi(u, u') = 0$ for each $u, u' \in U \setminus \{u^A\}$. Hence the output is 1 when reaching the accepting state u^A, and 0 otherwise. Such a binary reward model is common in the reward machine literature [8, 28].

3 MAPRL: Constructing Reward Machines from Partial-Order Plans

We can solve an MMDP $\mathcal{M}$ using existing MARL techniques (any inapplicable joint action can be modelled as an applicable action with no effect). However, since the size of the state and joint action space increases exponentially with the number of agents, this approach does not scale well. Instead, we propose a novel approach which assumes that the underlying MAP $\mathcal{D}, \mathcal{P}$ is known a priori. The idea is to use the solution of the MAP to guide reinforcement learning, similar to previous work in the single-agent setting [11].

3.1 Multiagent Reinforcement Learning Formulation

We assume that the MMDP $\mathcal{M}$ that we want to solve has a certain structure that we can exploit. Concretely, there exists a MAP $\mathcal{D} =$

$\langle \Psi, \mathcal{A} \rangle$, $\mathcal{P} = \langle N, \mathcal{C}, I, G \rangle$ that models the concurrent behavior of agents in the MMDP $\mathcal{M}$. In addition, each agent $n \in [\![N]\!]$ has an individual agent MDP $\mathcal{M}_n = \langle S_n, A_n^1, P_n, R_n, \mathcal{L}_n \rangle$ that governs the local dynamics of agent n. Here, $\mathcal{L}_n : S_n \to 2^{F_n}$ is a label function that maps states in S_n to planning states on the fluents of n in the MAP $\mathcal{D}, \mathcal{P}$. In what follows we describe how the MMDP $\mathcal{M} = \langle N, S, \{A_n\}_{n \in [\![N]\!]}, P, R \rangle$ that we want to solve is composed of the MAP $\mathcal{D}, \mathcal{P}$ and the agent MDPs $\mathcal{M}_n$, $n \in [\![N]\!]$.

Given the MAP $\mathcal{D}, \mathcal{P}$, let $O_{pub} = \{o \in O : (\text{Add}(o) \cup \text{Del}(o)) \cap F_{pub} \neq \emptyset\}$ be the set of public operators with at least one public effect. The state space of $\mathcal{M}$ is factored as $S = S_1 \times \cdots \times S_N \times S_{pub}$, where S_n, $n \in [\![N]\!]$, is the state space of the agent MDP $\mathcal{M}_n$ and $S_{pub} = 2^{F_{pub}}$ is derived from the public fluents of $\mathcal{P}$. The action set of agent n is $A_n = A_n^1 \cup A_n^2$, where A_n^1 is the action set of the agent MDP $\mathcal{M}_n$ that acts only on S_n, while $A_n^2 \subseteq O_{pub}$ are the public operators of n in $\mathcal{P}$.

We next describe how the transition kernel P of $\mathcal{M}$ is defined. Given a state $s = (s_1, \ldots, s_N, s_{pub})$ and a joint action $a = (a_1, \ldots, a_N)$, we first identify the set of agents $\mathcal{I} = \{n \in [\![N]\!] : a_n \in A_n^2\}$ that apply public operators in a. We next define a joint planning operator $a^2 = \bigtimes_{n \in \mathcal{I}} a_n \times \bigtimes_{n \in [\![N]\!] \setminus \mathcal{I}} \texttt{noop}(n)$ in which the action of each agent $n \in [\![N]\!] \setminus \mathcal{I}$ is replaced by a no-op operator. We also define a planning state $s_{\mathcal{I}} = s_{pub} \times \bigtimes_{n \in \mathcal{I}} \mathcal{L}_n(s_n)$ restricted to fluents in $F_{pub} \cup \bigcup_{n \in \mathcal{I}} F_n$. To determine the probability of a transition (s, a, s'), we need to know whether or not s' satisfies the effects of the joint planning operator a^2 in the planning state $s_{\mathcal{I}}$.

According to the definition of the MAP $\mathcal{D}, \mathcal{P}$, the joint operator a^2 is applicable in state $s_{\mathcal{I}}$ if $\text{Pre}(a^2) \subseteq s_{\mathcal{I}}$, and results in a new state $s'_{\mathcal{I}} = s_{\mathcal{I}} \oplus a^2$. From the perspective of the MAP $\mathcal{D}, \mathcal{P}$, the probability of transitioning to a state $s' = (s'_1, \ldots, s'_N, s'_{pub})$ is 1 if and only if the associated planning state $s'_{pub} \times \bigtimes_{n \in \mathcal{I}} \mathcal{L}_n(s'_n)$ equals $s'_{\mathcal{I}}$, else it is 0. On the other hand, the transition probability of each agent $n \in [\![N]\!] \setminus \mathcal{I}$ is governed by the agent MDP $\mathcal{M}_n$. Hence we can express the transition kernel P of the MMDP $\mathcal{M}$ as follows:

$$P(s'|s,a) = \begin{cases} 0, \text{ if } \text{Pre}(a^2) \not\subseteq s_{\mathcal{I}} \text{ or } s'_{pub} \times \bigtimes_{n \in \mathcal{I}} \mathcal{L}_n(s'_n) \neq s'_{\mathcal{I}}, \\ 1 \cdot \prod_{n \in [\![N]\!] \setminus \mathcal{I}} P_n(s'_n | s_n, a_n), \text{ otherwise.} \end{cases}$$

Hence the public part of joint actions is deterministic, while private actions may not be. The reward $R(s, a)$ is 1 when the goal condition is satisfied in s, i.e. $G \subseteq s_{pub} \times \bigtimes_{n \in [\![N]\!]} \mathcal{L}_n(s_n)$, and 0 otherwise.

Example 2. *Let $\mathcal{D}, \mathcal{P}$ be the MAP defined in Example 1, and let $\mathcal{M}$ be a corresponding MMDP. Even though $\mathcal{P}$ has a private operator* $\texttt{move}(1, a, c)$*, in the MMDP $\mathcal{M}$ agent 1 may have to apply multiple stochastic actions to move from a to c. However, since* $\texttt{push}(1, r, a, b)$ *is a public operator, b has to be reachable from a in one step as before.*

3.2 Overview of MAPRL Algorithm

Our proposed algorithm, Multiagent Planning for Reinforcement Learning (MAPRL)[1], solves a given MMDP $\mathcal{M}$ by exploiting the underlying structure as follows:

1. Translate the underlying MAP $\mathcal{D}, \mathcal{P}$ to a single-agent planning problem $\mathcal{D}', \mathcal{P}'$ [4].
2. Solve the single-agent problem using FMAP [29] to obtain a partial-order plan (POP) π_{POP}.
3. Use the POP π_{POP} to construct a reward machine (RM) $\mathcal{R}_n$ for each agent $n \in [\![N]\!]$.
4. Learn a policy π_n for each agent $n \in [\![N]\!]$ by solving the agent MDP $\mathcal{M}_n$ guided by the RM $\mathcal{R}_n$.

[1] The code is publicly available at https://github.com/Alee08/maprl.

3.3 Constructing Reward Machines

Here we explain how to construct a reward machine $\mathcal{R}_n$ for each agent $n \in [\![N]\!]$ given a POP π_{POP} that solves the single-agent problem $\mathcal{D}', \mathcal{P}'$ induced by the MAP $\mathcal{D}, \mathcal{P}$. Recall that $\pi_{POP} = \langle \mathcal{G}, L \rangle$ consists of an acyclic graph $\mathcal{G} = (V, E)$ and a set of causal links L, where $V = \{(o_I, 1), (o_G, 1)\} \cup V_O$, $V_O \subseteq O \times \mathbb{N}$, $E \subseteq V \times V$ and $L \subseteq V \times F \times V$.

The first step is to process all joint operators encoded in π_{POP}. Recall that joint operators are represented in $\mathcal{D}', \mathcal{P}'$ as subsequences of type $\langle \texttt{start-}X,\ X(i), \ldots, X(j), \texttt{end-}X \rangle$, which makes them easy to identify. Let $V_{\mathbf{o}} \subseteq V$ be the subset of operator instances associated with such a joint operator $\mathbf{o}$, and let $\overline{V} = V \setminus V_{\mathbf{o}}$. The result of processing $\mathbf{o}$ is a new POP $\pi'_{POP} = \langle (V', E'), L' \rangle$ such that $V' = \overline{V} \cup \{(\mathbf{o}, k)\}$, $E' = (E \cap (\overline{V} \times \overline{V})) \cup E_{\mathbf{o}}$ and $L' = (L \cap (\overline{V} \times F \times \overline{V})) \cup L_{\mathbf{o}}$, where $k \in \mathbb{N}$ is the smallest integer not yet used for $\mathbf{o}$. $E_{\mathbf{o}}$ contains an edge $(u, (\mathbf{o}, k))$ for each edge $(u, v) \in E \cap (\overline{V} \times V_{\mathbf{o}})$, and an edge $((\mathbf{o}, k), v)$ for each edge $(u, v) \in E \cap (V_{\mathbf{o}} \times \overline{V})$. The set $L_{\mathbf{o}}$ is similarly defined. Hence $V_{\mathbf{o}}$ is replaced by a single operator instance $(\mathbf{o}, k)$, and all edges and causal links to and from $(\mathbf{o}, k)$ are preserved. This procedure is repeated until all joint operators have been processed.

The next step is to perform a topological sort of the POP in order to obtain a sequential plan. Due to the properties of the POP, this sequential plan is guaranteed to solve the single-agent problem $\mathcal{D}', \mathcal{P}'$. This solution can then be translated back to a sequential plan π that solves the original MAP $\mathcal{D}, \mathcal{P}$ [4], by removing any extra predicates that were added to the translation $\mathcal{D}', \mathcal{P}'$. Each operator of π is either a simple operator (only a single agent acts) or a joint operator involving a subset of agents.

Given the solution π to the MAP $\mathcal{D}, \mathcal{P}$ and an agent $n \in [\![N]\!]$, let $\pi_n = \langle o_1, \ldots, o_k \rangle$ be the subsequence of π of simple or joint operators that involve agent n. The reward machine $\mathcal{R}_n = \langle U_n, \Sigma, u^0, u^A, \delta, \varphi \rangle$ contains one state u^i per operator o_i in the subsequence π_n, in addition to the initial state u^0, with $u^A \equiv u^k$. The set of symbols Σ equals 2^F, i.e. all subsets of fluents. For each $i \in [\![k]\!]$ and each state $s \in 2^F$, the transition function is defined as $\delta(u^{i-1}, s) = u^i$ if $\text{Pre}(o_i) \subseteq s$, and $\delta(u^{i-1}, s) = u^{i-1}$ otherwise. Hence a transition from u^{i-1} to u^i triggers in any state s that satisfies the precondition of o_i.

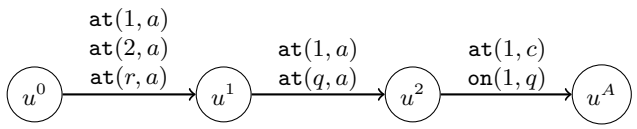
Consider two consecutive operators o_i, o_{i+1}, $i \in [\![k-1]\!]$, in the subsequence π_n. If o_i is a public operator in O_{pub} and the precondition of o_{i+1} satisfies $\text{Pre}(o_{i+1}) \subseteq (\text{Pre}(o_i) \setminus \text{Del}(o_i)) \cup \text{Add}(o_i)$, we can simplify the subsequence π_n by removing the operator o_{i+1}. This simplification is possible since the precondition of o_{i+1} is satisfied right after applying o_i, and results in a more compact RM $\mathcal{R}_n$ with fewer states. In addition, we can remove any static fluents that are never added or deleted, such as $\texttt{pen}(q)$ and $\texttt{box}(r)$ in Example 1.

Example 3. *Consider the MMDP in Example 2. An example operator sequence π solving the associated MAP is given by*

$$\pi = \langle (\texttt{push}(1, r, a, b), \texttt{push}(2, r, a, b)), \texttt{move}(1, b, a), \\ \texttt{pick}(1, q, a), \texttt{move}(1, a, c), \texttt{drop}(1, q, c) \rangle.$$

All operators in π are associated with agent 1, but the precondition $\texttt{at}(1, b)$ *of* $\texttt{move}(1, b, a)$ *is satisfied after applying the public joint operator* $(\texttt{push}(1, r, a, b), \texttt{push}(2, r, a, b))$,

and the precondition $\texttt{at}(1, a)$ *of* $\texttt{move}(1, a, c)$ *is satisfied after applying the public operator* $\texttt{pick}(1, q, a)$*. Hence* $\pi_1 = \langle(\texttt{push}(1, r, a, b), \texttt{push}(2, r, a, b)), \texttt{pick}(1, q, a), \texttt{drop}(1, q, c)\rangle$ *resulting in the following RM* $\mathcal{R}_1$ *for agent* 1.



3.4 Execution Model

Given the agent MDP $\mathcal{M}_n = \langle S_n, A_n^1, P_n, R_n, \mathcal{L}_n\rangle$ and the RM $\mathcal{R}_n = \langle U_n, \Sigma, u^0, u^A, \delta, \varphi\rangle$, agent $n \in [\![N]\!]$ learns an individual policy π_n^i for each RM state $u^i \in U_n \setminus \{u^A\}$. Concretely, we use QRM [28] to learn the policy of each agent, and apply reward shaping as is common in the RM literature. The policy π_n^i aims to satisfy the condition $\sigma \in 2^F$ on the only outgoing transition from u^i by reaching a state $s_n \in S_n$ such that $\mathcal{L}_n(s_n) \subseteq \sigma$. Since some conditions cannot be satisfied by agent n itself (e.g. $\texttt{at}(2, a)$ on the transition from u^0 to u^1 in the RM $\mathcal{R}_1$ from Example 3), once agent n has satisfied its own conditions, it has to wait for other agents to satisfy the remaining conditions. For this purpose, we assume that the agent MDP $\mathcal{M}_n$ contains a no-op action $a_{noop} \in A_n^1$ that always causes the agent to remain in the same state of S_n.

Our execution model assumes that agent n automatically performs the no-op action a_{noop} once it has reached a state $s_n \in S_n$ that satisfies its part of the condition σ on the outgoing transition from u^i. Once σ is fully satisfied by other agents, the outgoing transition triggers and the new RM state becomes u^{i+1}. In addition, if the operator o^{i+1} associated to u^{i+1} is a public operator, our transition model assumes that the deterministic operator o^{i+1} is *automatically applied*. Hence in RM state u^{i+1}, agent n is initially in a state $s'_n \in S_n$ such that $\mathcal{L}_n(s'_n) = ((\mathcal{L}_n(s_n) \setminus \text{Del}(o^{i+1})) \cup \text{Add}(o^{i+1})) \cap F_n$ is consistent with the effect of o^{i+1} on the fluents of n.

As a consequence of our execution model, we can solve an MMDP in a fully distributed manner once the RMs have been constructed, since the policy π_n of each agent $n \in [\![N]\!]$ only acts in the individual agent MDP $\mathcal{M}_n$. Any joint actions required to solve the problem are computed by the MAP planner and automatically applied in the corresponding RM states, and each agent n only has to learn to take actions in A_n^1 which satisfy its own conditions in the RM. The only coordination requirement on our execution model is that agents have to broadcast the high-level effects of their actions for other agents to be aware if a given condition is satisfied.

4 Experiments

In this section, we conduct an empirical evaluation of MAPRL across three domains. Each domain consists of three tasks with increasing complexity. The domains are variants of Office World [28], Maze [4], and Temple Quest, a novel domain inspired by classic exploration and puzzle-solving challenges. Each domain comprises three tasks with gradually increasing complexity. Specifically, we increase the complexity by adding more agents, including additional goal conditions, and/or expanding the map size in each successive task. The high-level planning description only considers the current room of the agent, not the precise location within each room, though we need a fluent of type $\texttt{at}(n, x)$ to represent whether an agent n is next to a certain object x inside a room. It is only possible to move between two rooms if the rooms are connected. The agent MDP $\mathcal{M}_n$ of each agent n contains actions for moving right, left, up and down.

Table 1: Training steps needed to achieve a near-optimal policy (lower is better). A value of 1 700 000 means that the algorithm timed out within the given budget. The best result in each row appears in bold.

Domain	Task	Agents	MAPRL	CQRM	IQL
Office World	1	2	**600**	21 000	1 700 000
	2	2	**900**	65 000	1 700 000
	3	5	**215000**	—	1 700 000
Concurrent Maze	1	8	**63000**	—	1 700 000
	2	8	**213000**	—	1 700 000
	3	8	**1167000**	—	1 700 000
Temple Quest	1	10	**461000**	—	1 700 000
	2	10	**465000**	—	1 700 000
	3	10	**575000**	—	1 700 000

We compare our MAPRL approach against two baseline algorithms which represent the two extremes of the distributed/centralized spectrum: IQL [27] and Centralized QRM (CQRM) with a single hand-crafted RM. Intermediate MARL algorithms such as value-factorisation (e.g. QMIX) are omitted because they collapse on strongly coupled tasks as previously demonstrated [19].

CQRM learns the joint policy $\pi : S \to \Delta(A)$ of an MMDP $\mathcal{M}$ over a space in which the subgoals of the agents are shared in the RM. In contrast, our algorithm MAPRL distributes learning, and each agent has its own MDP, policy and reward machine (RM). All algorithms use a discount factor of $\gamma = 0.9$ and a learning rate of $\alpha = 0.5$. Action selection follows a greedy exploration strategy with $\epsilon = 0.1$. The Q-table is initialized optimistically to encourage exploration. Each episode lasts 1000 steps and the optimal policy is evaluated every 100 episodes.

Table 1 shows the number of training steps needed for convergence to a near-optimal policy for each of the three compared algorithms. We highlight two key conclusions that we can draw from the table. First, MAPRL consistently requires orders of magnitude fewer training steps than both baselines, confirming that coupling the high-level plan with distributed learning dramatically speeds up convergence. Second, although the centralized baseline CQRM is able to solve the first two Office World tasks, it requires 35–70 times more samples than MAPRL and *fails completely* once the number of agents grows from two to five (Task 3). The absence of CQRM results for Concurrent Maze (8 agents) and Temple Quest (10 agents) is therefore *not* an omission but a reflection of its inability to scale: the joint Q-table grows exponentially with the agent count, making training and even storage infeasible beyond a handful of agents. Finally, IQL never converges within the time budget in any setting, underscoring that independent learning is insufficient in these tightly-coupled domains. Overall, the results support our claim that MAPRL's blend of partial-order planning and reward-machine guidance is essential for sample-efficient learning in strongly coupled multi-agent problems of realistic size.

4.1 Concurrent Office World

The Office World domain extends the original Office World [28] to multiple agents and concurrent actions. There are five agents, distributed as two managers and three employees. Managers, in addition to movement, have actions for accessing server rooms via a private entrance. Employees, in addition to movement, have actions for accessing server rooms via a separate public entrance. To use a private entrance, both managers must simultaneously unlock the door for security reasons (i.e. the affordance is $[2, 2]$). Employees can enter

a public entrance freely (affordance $[1, 3]$) but once any employee enters, the door closes permanently, preventing further access. The goal is for all managers and employees to enter the server room; both managers and employees must synchronize to enter together.

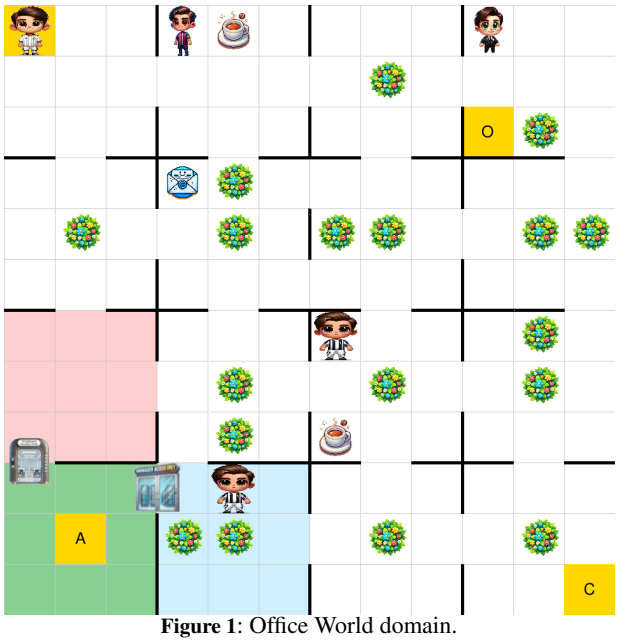


Figure 1: Office World domain.

Figure 1 illustrates the domain, with the server room marked by A and managers depicted in black-and-white uniforms, while the three employees start at the top of the grid. The planner learns that for all agents to reach the server room, the managers and employees have to enter the server room together, respectively. When both managers reach a designated "blue" area next to the private entrance, they automatically take the joint action together to enter the server room. A similar synchronization mechanism applies to the employees, who are guided to reach a "red" area next to the public entrance. At that point, the employees automatically take the joint action to enter the server room together. Immediately after entering, the entrance permanently closes behind them.

Task 1 is carried out solely by the managers, who must coordinate in order to arrive at the server room. Task 2 is also performed by the managers, but in this case they first go and collect the nearest coffee, and then one manager proceeds to room B while the other goes to room C. Task 3 is undertaken by both managers and employees: everyone must first pick up the coffee closest to them and deliver it to O, B, and C, and only then must they coordinate to reach the server room A.

In Figures 2-3, we compared our MAPRL approach against CQRM and IQL. CQRM can solve the first two tasks, but when we increase the number of agents in Task 3, applying CQRM is unfeasible due to the exponential action space. IQL fails to solve all tasks since the agents act independently and never learn to coordinate. We believe that IQL is representative of all cooperative MARL algorithms that distribute learning. In contrast, MAPRL converges in under 100 episodes thanks to the high-level plan and joint execution model.

4.2 *Concurrent Maze*

Figure 4 illustrates our variant of Maze, in which eight agents must explore and exit the labyrinth while avoiding the holes in the floor.

Figure 2: Two-agent Task 1 - Concurrency Office World 12x12.

Figure 3: Five-agent Task 3 - Concurrency Office World 12x12.

Six agents (in red or black) are climbers skilled at traversing dangerous bridges, while two are mariners specialized in rowing to cross rivers. Bridges can only be crossed once before collapsing, requiring the climbers to coordinate. Similarly, mariners must work in pairs to traverse rivers.

In Task 1, the three red climbers must reach point A, while the three black climbers must reach point E. To do so the climbers must coordinate to cross critical bridges together. The two mariners in black-and-white striped shirts also have to reach point E, requiring them to row concurrently. Task 2 extends Task 1 by requiring the mariners to first collect oars near their starting location, while the climbers must pick up torches to safely cross bridges in the dark. In Task 3, the mariners must also visit area A, while the climbers need to retrieve a scroll in the red zone bordered by two bridges: to acquire the scroll, all six climbers must cross the two bridges together.

As shown in Figure 5, IQL fails to solve the three tasks, whereas MAPRL completes them quickly, including the more challenging task with multiple concurrent actions.

4.3 *Temple Quest*

The third domain we present is called *Temple Quest*. In this domain (Figure 6), ten explorers must coordinate to explore various locations and reach hidden treasures after obtaining the keys necessary to unlock them. There are six explorers specialized in opening the temple's stone doors (the three at the top-left in the figure, row 0, and the three wearing red shirts), and four explorers responsible for moving large boulders that block passageways (two wearing black-and-white striped shirts and two wearing green shirts). Once a stone door is unlocked and crossed, it is destroyed; to move a boulder located

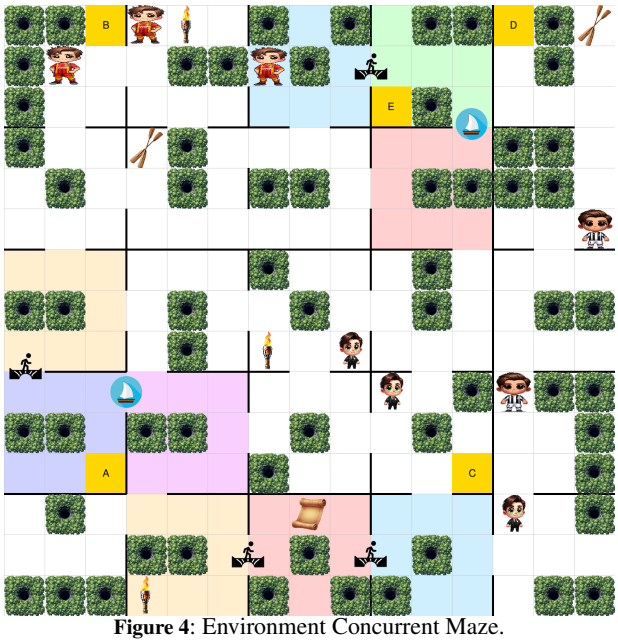


Figure 4: Environment Concurrent Maze.

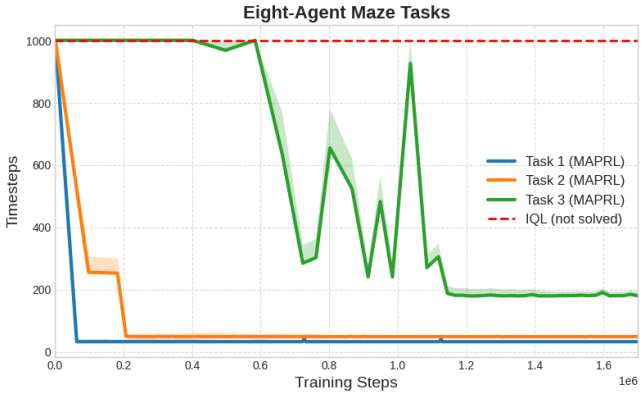


Figure 5: Eight-agent - Concurrency Maze 15x15.

between temple columns, two explorers must collaborate; otherwise, the boulder remains immovable.

In Task 1, the three red-shirt explorers coordinate when they reach the light-green room. Upon approaching the stone door, they coordinate to open the door and gain access to the dark-green room. They proceed similarly for the door connecting the light-red room to the dark-red room, which holds the treasure. Meanwhile, the green-shirt agents move the boulder between the two columns blocking passage from the light-yellow room to the dark-yellow room, allowing them to reach the treasure. Likewise, the remaining five agents must coordinate to move another boulder (in the light purple and dark purple room); these are the scouts wearing black-and-white striped shirts. Finally, the three scouts in the top row of the map unlock the temple door (which connects the light blue area to the dark purple area).

Task 2 requires the agents to move from the temple to the center of the map, then pick up one of the keys, and finally coordinate with the other agents (as they did in Task 1) to reach the treasure. In Task 3, the six explorers responsible for opening stone doors must explore area B on the map. To do so, they must wait together in front of the door, enter the burgundy-colored B room simultaneously, and then all exit together through the second stone door to avoid leaving anyone trapped inside after the door collapses. Meanwhile, the other four explorers must move the boulders twice in order to enter and then

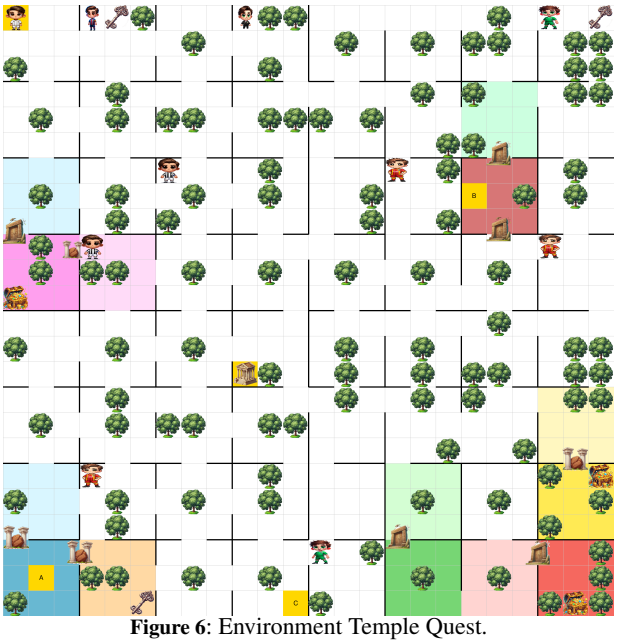


Figure 6: Environment Temple Quest.

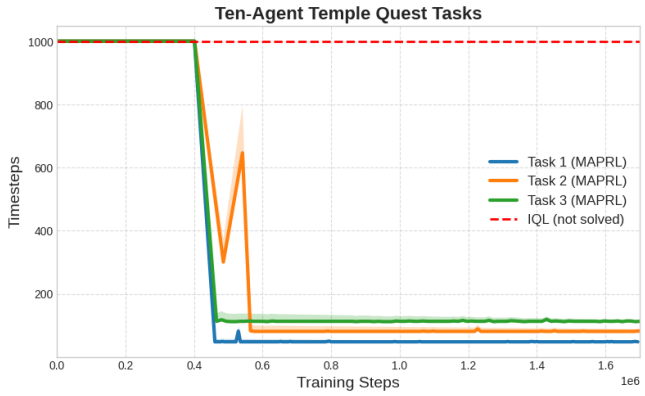


Figure 7: Ten-agent - Temple Quest 24x24.

exit A (the blue-colored room).

As illustrated in Figure 7, MAPRL consistently identifies the optimal policy within approximately 500,000 training steps, whereas IQL is unable to achieve the same outcome.

5 Conclusions

We present a novel approach for cooperative MARL in which joint actions are strongly coupled. Our approach computes a partial-order plan (POP) that solves a high-level MAP encoding all concurrent problem aspects. We use the POP to construct one RM per agent, allowing agents to train individual policies in a distributed manner. Experiments show that ours is the only centralized MARL approach that scales in complex domains to more than two or three agents.

Possible directions for future work include constructing the RMs directly from the acyclic graph of the POP, which would likely involve incorporating additional fluents to model unlabelled precedence edges of the graph, or constructing more expressive RMs that are not always linear. To obtain alternative courses of action, it would likely be necessary to compute a set of diverse POPs that solve the MAP in different ways. This would be more costly than computing a single POP, but may produce better quality RMs. Other ideas for future work are to consider stochastic joint actions and exploit the structure of problems in which agents have similar agent MDPs.

Acknowledgements

Alessandro Trapasso is partially supported by the PNRR MUR project PE0000013-FAIR. Anders Jonsson is partially supported by Spanish grants PID2023-147145NBI00 and MCIN/AEI/10.13039/501100011033 under the Maria de Maeztu Units of Excellence Programme (CEX2021-001195-M).

References

[1] C. Amato. A First Introduction to Cooperative Multi-Agent Reinforcement Learning, 2024.

[2] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588:77–82, 2020.

[3] C. Boutilier and R. Brafman. Partial-Order Planning with Concurrent Interacting Actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001.

[4] M. Crosby, A. Jonsson, and M. Rovatsos. A Single-Agent Approach to Multi-agent Planning. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 237–242, 2014.

[5] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de las Casas, C. Donner, L. Fritz, C. Galperti, A. Huber, J. Keeling, M. Tsimpoukelli, J. Kay, A. Merle, J.-M. Moret, S. Noury, F. Pesamosca, D. Pfau, O. Sauter, C. Sommariva, S. Coda, B. Duval, A. Fasoli, P. Kohli, K. Kavukcuoglu, D. Hassabis, and M. Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602:414–419, 2022.

[6] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 2974–2982, 2018.

[7] D. Furelos-Blanco and A. Jonsson. Solving Multiagent Planning Problems with Concurrent Conditional Effects. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 7594–7601, 2019.

[8] D. Furelos-Blanco, M. Law, A. Jonsson, K. Broda, and A. Russo. Induction and Exploitation of Subgoal Automata for Reinforcement Learning. *Journal of Artificial Intelligence Research*, 70, 2021.

[9] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[10] C. Guestrin, D. Koller, and R. Parr. Multiagent Planning with Factored MDPs. In *Neural Information Processing Systems (NeurIPS)*, pages 1523–1530, 2001.

[11] L. Illanes, X. Yan, R. Toro-Icarte, and S. A. McIlrath. Symbolic Plans as High-Level Instructions for Reinforcement Learning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 540–550, 2020.

[12] D. L. Kovacs. A Multi-Agent Extension of PDDL3.1. In *Proceedings of the 3rd Workshop on the International Planning Competition (IPC)*, pages 19–27, 2012.

[13] N. Lazic, T. Lu, C. Boutilier, M. Ryu, E. Wong, B. Roy, and G. Imwalle. Data center cooling using model-predictive control. In *Advances in Neural Information Processing Systems*, 2018.

[14] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Neural Information Processing Systems (NeurIPS)*, 2017.

[15] S. Maliah, R. Brafman, and G. Shani. Increased Privacy with Reduced Communication in Multi-Agent Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 209–217, 2017.

[16] C. Neary, Z. Xu, B. Wu, and U. Topcu. Reward Machines for Cooperative Multi-Agent Reinforcement Learning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 934–942, 2021.

[17] R. Nissim and R. Brafman. Distributed Heuristic Forward Search for Multi-agent Planning. *Journal of Artificial Intelligence Research*, 51: 293–332, 2014.

[18] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. In *Neural Information Processing Systems*, 2022.

[19] N. Prabhakar, R. Singh, H. Kokel, S. Natarajan, and P. Tadepalli. Combining planning and reinforcement learning for solving relational multiagent domains, 2025. URL https://arxiv.org/abs/2502.19297.

[20] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson. Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. *Journal of Machine Learning Research*, 21:1–51, 2020.

[21] E. D. Sacerdoti. The non-linear nature of plans. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 206–214, 1975.

[22] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature*, 588:604–609, 2020.

[23] S. Shekhar and R. Brafman. Representing and planning with interacting actions and privacy. *Artificial Intelligence*, 278:103200, 2020.

[24] R. Singh, N. Prabhakar, S. Natarajan, and P. Tadepalli. Exploiting Relational Planning and Task-Specific Abstractions for Multiagent Reinforcement Learning in Relational Domains. In *Cooperative Multi-Agent Systems Decision-making and Learning Workshop at AAAI*, 2024.

[25] M. Štolba, A. Komenda, and D. L. Kovacs. Competition of Distributed and Multiagent Planners (CoDMAP). In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 4343–4345, 2016.

[26] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel. Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward. In *Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 2085–2087, 2018.

[27] M. Tan. Multi-agent reinforcement learning: independent versus cooperative agents. In *Proceedings of the Tenth International Conference on International Conference on Machine Learning*, ICML'93, page 330–337, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. ISBN 1558603077.

[28] R. Toro-Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlrath. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. *Journal of Artificial Intelligence Research*, 73:173–208, 2022.

[29] A. Torreño, Ó. Sapena, and E. Onaindia. Global Heuristics for Distributed Cooperative Multi-Agent Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 225–233, 2015.

[30] G. Varricchione, N. Alechina, M. Dastani, and B. Logan. Synthesising Reward Machines for Cooperative Multi-Agent Reinforcement Learning. In *Proceedings of the European Conference on Multi-Agent Systems (EUMAS)*, pages 328–344, 2023.

[31] J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang. QPLEX: Duplex dueling multi-agent Q-learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[32] D. S. Weld. An Introduction to Least Commitment Planning. *AI Magazine*, 15(4):27–61, 1994.

[33] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games. In *Neural Information Processing Systems (NeurIPS)*, 2022.