

Metadata of the chapter that will be visualized in SpringerLink

Book Title	Autonomous Agents and Multiagent Systems. Best and Visionary Papers	
Series Title		
Chapter Title	Learning Reward Machines in Cooperative Multi-agent Tasks	
Copyright Year	2024	
Copyright HolderName	The Author(s), under exclusive license to Springer Nature Switzerland AG	
Corresponding Author	Family Name	Ardon
	Particle	
	Given Name	Leo
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	Imperial College London
	Address	London, UK
	Email	leo.ardon19@imperial.ac.uk
	ORCID	http://orcid.org/0000-0003-4400-7127
Author	Family Name	Furelos-Blanco
	Particle	
	Given Name	Daniel
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	Imperial College London
	Address	London, UK
	Email	d.furelos-blanco18@imperial.ac.uk
	ORCID	http://orcid.org/0000-0001-7461-1910
Author	Family Name	Russo
	Particle	
	Given Name	Alessandra
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	Imperial College London
	Address	London, UK
	Email	a.russo@imperial.ac.uk
	ORCID	http://orcid.org/0000-0002-3318-8711
Abstract	This paper presents a novel approach to Multi-Agent Reinforcement Learning (MARL) that combines cooperative task decomposition with the learning of <i>Reward Machines (RMs)</i> encoding the structure of the	

sub-tasks. The proposed method helps deal with the non-Markovian nature of the rewards in partially observable environments and improves the interpretability of the learnt policies required to complete a cooperative task. The RMs associated with the sub-tasks are learnt in a decentralised manner and then used to guide the behaviour of each agent in a team acting towards a common goal. By doing so, the complexity of a cooperative multi-agent problem is reduced, allowing for more effective learning. The results suggest that our approach is a promising direction for future research in cooperative MARL, especially in complex and partially observable environments.

Keywords (separated by '-')	Multi-Agent Reinforcement Learning - Reward Machines - Neuro-Symbolic - Symbolic Machine Learning
--------------------------------	---



Learning Reward Machines in Cooperative Multi-agent Tasks

Leo Ardon^(✉), Daniel Furelos-Blanco, and Alessandra Russo

Imperial College London, London, UK

{leo.ardon19,d.furelos-blanco18,a.russo}@imperial.ac.uk

Abstract. This paper presents a novel approach to Multi-Agent Reinforcement Learning (MARL) that combines cooperative task decomposition with the learning of *Reward Machines (RMs)* encoding the structure of the sub-tasks. The proposed method helps deal with the non-Markovian nature of the rewards in partially observable environments and improves the interpretability of the learnt policies required to complete a cooperative task. The RMs associated with the sub-tasks are learnt in a decentralised manner and then used to guide the behaviour of each agent in a team acting towards a common goal. By doing so, the complexity of a cooperative multi-agent problem is reduced, allowing for more effective learning. The results suggest that our approach is a promising direction for future research in cooperative MARL, especially in complex and partially observable environments.

AQ1

Keywords: Multi-Agent Reinforcement Learning · Reward Machines · Neuro-Symbolic · Symbolic Machine Learning

1 Introduction

With impressive advances in the past decade [12, 20, 24], the Reinforcement Learning (RL) paradigm [26] appears as a promising avenue in the quest to have machines learn autonomously how to achieve a goal. Originally evaluated in the context of a single agent interacting with the rest of the world, researchers have since then extended the approach to the multi-agent setting (MARL) [2, 23, 25], where multiple autonomous entities learn in the same environment. Multiple agents learning concurrently brings a set of challenges, such as partial observability, non-stationarity, and scalability [1, 3]; but, it also represents a more realistic setting of the problem to solve. Like in many real-life scenarios, the actions of one individual often affect the way others act and should therefore be considered in the learning process.

A sub-field of MARL called Cooperative Multi-Agent Reinforcement Learning (CMARL) focuses on learning policies for multiple agents that must coordinate their actions to achieve a shared objective. These tasks can be tackled using ‘divide and conquer’ strategies; that is, by decomposing them into smaller and simpler tasks assigned to different agents acting in parallel in the environment.

The problem of learning an optimal sequence of actions now also includes the need to strategically divide the task among agents that must learn to coordinate and communicate in order to find the optimal way to cooperatively accomplish the goal.

An emergent field of research in the RL community exploits finite-state machines to encode the structure of the reward function; ergo, the structure of the task at hand. This new construct called *Reward Machine* (RM) [27] can model non-Markovian reward and provides a symbolic interpretation of the stages required to complete a task. While RMs can be handcrafted, it is often impractical to do so. Hence, learning RMs has been the object of several works in recent years [6, 13, 14, 17, 29, 30] as a way to reduce human input. However, the aforementioned challenges in the multi-agent setting hinder the learning of an RM encoding the global task. With multiple agents at play, the number of states of the global RM grows exponentially, making its learning extremely challenging.

In this paper, we argue that learning RMs for smaller tasks obtained from an appropriate task decomposition is more efficient. The global RM can later be reconstructed through a parallel composition of the learned RMs for the sub-tasks. Inspired by previous work on using RMs to decompose and distribute a global task to a team of collaborative agents [21], we propose a method that combines task decomposition with the autonomous learning of RMs in a multi-agent setting. Our approach offers a more scalable and interpretable way to learn the structure of a complex task involving collaboration among multiple agents. We present an algorithm that learns (in parallel) the RM for each agent’s sub-task and the associated RL policies that enable agents to achieve the global cooperative task. We compare our approach with (i) a method employing handcrafted RMs [21], and (ii) a decentralized Q-learning approach that does not leverage the task’s structure. We empirically show in two existing environments [21] that with our method a team of agents is able to learn how to solve a collaborative task more efficiently (i.e. with fewer training iterations) than without using RMs. The policies trained with our approach also achieve similar performance than with handcrafted RMs while requiring less human input, thus demonstrating the value of the proposed technique. Finally, the RMs learnt with our algorithm are easily interpretable and accurately characterize the structure of the task at hand.

2 Background

Given a finite set $\mathcal{X}$, $\Delta(\mathcal{X})$ denotes the probability simplex over $\mathcal{X}$, $\mathcal{X}^*$ denotes (possibly empty) sequences of elements from $\mathcal{X}$, and $\mathcal{X}^+$ is a non-empty sequence. The symbols $\perp$ and $\top$ denote false and true, respectively.

2.1 Reinforcement Learning

We consider T -*episodic labeled* Markov decision processes (MDPs) [30], characterized by a tuple $\langle \mathcal{S}, s_I, \mathcal{A}, p, r, T, \gamma, \mathcal{P}, l \rangle$ consisting of a set of states $\mathcal{S}$, an initial state $s_I \in \mathcal{S}$, a set of actions $\mathcal{A}$, a transition function $p : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$, a *not* necessarily Markovian reward function $r : (\mathcal{S} \times \mathcal{A})^+ \times \mathcal{S} \rightarrow \mathbb{R}$, the time horizon T of each episode, a discount factor $\gamma \in [0, 1)$, a finite set of propositions $\mathcal{P}$, and a labeling function $l : \mathcal{S} \times \mathcal{S} \rightarrow 2^{\mathcal{P}}$.

A (state-action) *history* $h_t = \langle s_0, a_0, \dots, s_t \rangle \in (\mathcal{S} \times \mathcal{A})^* \times \mathcal{S}$ is mapped into a *label trace* (or *trace*) $\lambda_t = \langle l(s_0, s_1), \dots, l(s_{t-1}, s_t) \rangle \in (2^{\mathcal{P}})^+$ by applying the labeling function to each state transition in h_t . We assume that the reward function can be written in terms of a label trace, i.e. formally $r(h_{t+1}) = r(\lambda_{t+1}, s_{t+1})$. The goal is to find an optimal *policy* $\pi : (2^{\mathcal{P}})^+ \times \mathcal{S} \rightarrow \mathcal{A}$, mapping (traces-states) to actions, in order to maximize the expected cumulative discounted reward (or return) $R_t = \mathbb{E}_{\pi} \left[\sum_{k=t}^T \gamma^{k-t} r(\lambda_{k+1}, s_{k+1}) \right]$.

At time t , the agent keeps a trace $\lambda_t \in (2^{\mathcal{P}})^+$, and observes state $s_t \in \mathcal{S}$. The agent chooses the action to execute with its policy $a = \pi(\lambda_t, s_t) \in \mathcal{A}$, and the environment transitions to state $s_{t+1} \sim p(\cdot \mid s_t, a_t)$. As a result, the agent observes a new state s_{t+1} and a new label $\mathcal{L}_{t+1} = l(s_t, s_{t+1})$, and gets the reward $r_{t+1} = r(\lambda_{t+1}, s_{t+1})$. The trace $\lambda_{t+1} = \lambda_t \oplus \mathcal{L}_{t+1}$ is updated with the new label.

In this work, we focus on the *goal-conditioned* RL problem [18] where the agent's task is to reach a goal state as rapidly as possible. We thus distinguish between two types of traces: *goal* traces and *incomplete* traces. A trace λ_t is said to be a *goal* trace if the task's goal has been achieved before t otherwise we say that the trace is *incomplete*. Formally, the MDP includes a *termination function* $\tau : (2^{\mathcal{P}})^+ \times \mathcal{S} \rightarrow \{\perp, \top\}$ indicating that the trace λ_t at time t is an *incomplete* trace if $\tau(\lambda_t, s_t) = \perp$ or a *goal* trace if $\tau(\lambda_t, s_t) = \top$. We assume the reward is 1 for goal traces and 0 otherwise. For ease of notation, we will refer to a goal trace as $\lambda_t^{\top}$ and an incomplete trace as $\lambda_t^{\perp}$. Note that we assume the agent-environment interaction stops when the task's goal is achieved. This is without loss of generality as it is equivalent to having a final absorbing state with a null reward associated and no label returned by the labelling function l .

The single-agent framework above can be extended to the collaborative *multi-agent* setting as a Markov game. A cooperative *Markov game* of N agents is a tuple $\mathcal{G} = \langle N, \mathcal{S}, s_I, \mathcal{A}, p, r, \tau, T, \gamma, \mathcal{P}, l \rangle$, where $\mathcal{S} = \mathcal{S}_1 \times \dots \times \mathcal{S}_N$ is the set of joint states, $s_I \in \mathcal{S}$ is a joint initial state, $\mathcal{A} = \mathcal{A}_1 \times \dots \times \mathcal{A}_N$ is the set of joint actions, $p : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is a joint transition function, $r : (\mathcal{S} \times \mathcal{A})^+ \times \mathcal{S} \rightarrow \mathbb{R}$ is the collective reward function, $\tau : (2^{\mathcal{P}})^+ \times \mathcal{S} \rightarrow \{\perp, \top\}$ is the collective termination function, and $l : \mathcal{S} \times \mathcal{S} \rightarrow 2^{\mathcal{P}}$ is an event labeling function. The elements T , γ , and $\mathcal{P}$ are defined as for MDPs. Like Neary et al. [21], we assume each agent A_i 's dynamics are independently governed by local transition functions $p_i : \mathcal{S}_i \times \mathcal{A}_i \rightarrow \Delta(\mathcal{S}_i)$, hence the joint transition function is $p(s' \mid s, a) = \prod_{i=1}^N p(s'_i \mid s_i, a_i)$ for all $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$. The objective is to find a team policy $\pi : (2^{\mathcal{P}})^+ \times \mathcal{S} \rightarrow \mathcal{A}$ mapping pairs of traces and joint states to joint actions that maximises the expected cumulative collective reward.

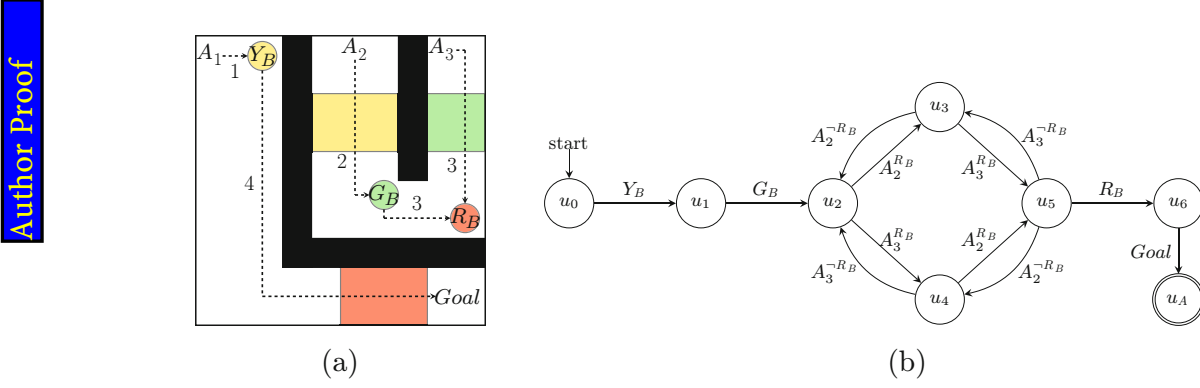


Fig. 1. Illustration of the THREEBUTTONS grid (a) and a RM modeling the task's structure (b) [21]. (Color figure online)

Example 1. We use the THREEBUTTONS task [21], illustrated in Fig. 1a, as a running example. The task consists of three agents (A_1 , A_2 , and A_3) that must cooperate for agent A_1 to get to the *Goal* position. To achieve this, they must open doors that prevent other agents from progressing by pushing buttons Y_B , G_B , and R_B . Agents can go UP, DOWN, LEFT, RIGHT or DO NOTHING. An agent starts pushing a button when it steps on the button's position; likewise, an agent stops pushing a button when it leaves the button's position. The button of a given colour opens the door of the same colour. Two agents are required to push R_B at once, involving synchronization. Once a door has been opened, it remains open until the end of the episode. The order in which high-level actions must be performed is shown in the figure: (1) A_1 pushes Y_B , (2) A_2 pushes G_B , (3) A_2 and A_3 push R_B , and (4) A_1 reaches *Goal*. We can formulate this problem by defining a shared reward function returning 1 when the *Goal* position is reached. The states solely consist of the position of the agents in the environment. The *proposition set* in this task is $\mathcal{P} = \{R_B, Y_B, G_B, A_2^{-R_B}, A_2^{R_B}, A_3^{-R_B}, A_3^{R_B}, Goal\}$, where (i) R_B , Y_B and G_B indicate that the red, yellow and green buttons have been pushed respectively, (ii) $A_i^{R_B}$ (resp. $A_i^{-R_B}$), where $i \in \{2, 3\}$, indicates that agent A_i has started (resp. has stopped) pushing the red button R_B , and (iii) *Goal* indicates that the goal position has been reached. A *goal trace* example is $\langle \{\}, \{Y_B\}, \{G_B\}, \{A_2^{R_B}\}, \{A_3^{R_B}\}, \{R_B\}, \{Goal\} \rangle$.

2.2 Reward Machines

We here introduce Reward Machines, the formalism we use to express decomposable (multi-agent) tasks.

Definitions. A Reward Machine [27, 28] is a finite-state machine that represents the reward function of an RL task. Formally, an RM is a tuple $M = \langle \mathcal{U}, \mathcal{P}, u_0, u_A, \delta_u, \delta_r \rangle$ where $\mathcal{U}$ is a set of states; $\mathcal{P}$ is a set of propositions; $u_0 \in \mathcal{U}$ is the initial state; $u_A \in \mathcal{U}$ is the final state; $\delta_u : \mathcal{U} \times 2^{\mathcal{P}} \rightarrow \mathcal{U}$ is a state-transition function such that $\delta_u(u, \mathcal{L})$ is the state that results from observing label $\mathcal{L} \in 2^{\mathcal{P}}$ in state $u \in \mathcal{U}$; and $\delta_r : \mathcal{U} \times \mathcal{U} \rightarrow \mathbb{R}$ is a reward-transition function such that

$\delta_r(u, u')$ is the reward obtained for transitioning from state $u \in \mathcal{U}$ to $u' \in \mathcal{U}$. We assume that (i) there are no outgoing transitions from u_A , and (ii) $\delta_r(u, u') = 1$ if $u' = u_A$ and 0 otherwise (following the reward assumption in Sect. 2.1). Given a trace $\lambda = \langle \mathcal{L}_0, \dots, \mathcal{L}_n \rangle$, a *traversal* is a sequence of RM states $\langle v_0, \dots, v_{n+1} \rangle$ such that $v_0 = u_0$, and $v_{i+1} = \delta_u(v_i, \mathcal{L}_i)$ for $i \in \{1, \dots, n\}$.

Ideally, RMs are such that (i) the cross product $\mathcal{S} \times \mathcal{U}$ of their states with the MDPs' make the reward and termination functions Markovian, and (ii) traversals for goal traces end in the final state u_A , and traversals for incomplete traces do not.

Example 2. Figure 1b shows an RM for the THREEBUTTONS task. For simplicity, edges are labelled using the single proposition that triggers the transitions instead of sets. The goal trace from Example 1 is associated with the traversal $\langle u_0, u_1, u_2, u_3, u_5, u_6, u_A \rangle$.

RL Algorithm. The Q-learning for RMs (QRM) algorithm [27, 28] exploits the task structure modelled through RMs. QRM learns a Q-function $q_u : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ for each state $u \in \mathcal{U}$ in the RM, which estimates the expected return from a given state-action pair. Given an experience tuple $\langle s, a, s' \rangle$, a Q-function q_u is updated according to the Bellman update as follows:

$$q_u(s, a) = q_u(s, a) + \alpha \left(\delta_r(u, u') + \gamma \max_{a'} q_{u'}(s', a') - q_u(s, a) \right),$$

where $u' = \delta_u(u, l(s, s'))$. All Q-functions (or a subset of them) of the RM are updated at each step using the same experience tuple in a counterfactual manner, by evaluating the effect of the experience tuple in all the states of the RM. QRM is guaranteed to converge to an optimal policy in the tabular case.

Multi-agent Decomposition. Neary et al. [21] recently proposed to decompose the RM for a multi-agent task into several RMs (one per agent) executed in parallel. The task decomposition into individual and independently learnable sub-tasks addresses the problem of non-stationarity inherent in the multi-agent setting. In addition, the use of RMs gives the high-level structure of the task each agent ought to solve. In this setting, each agent A_i has its own RM M_i , local state space $\mathcal{S}_i$ (e.g., it solely observes its position in the grid), and propositions $\mathcal{P}_i$ such that $\mathcal{P} = \bigcup_i \mathcal{P}_i$. Besides, instead of having a single opaque labelling function, each agent A_i employs its own labelling function $l_i : \mathcal{S}_i \times \mathcal{S}_i \rightarrow 2^{\mathcal{P}_i}$. Each labelling function is assumed to return at most one proposition per agent per timestep, and they should together output the same label as the global labelling function l .

Given a global RM M modelling the structure of the task at hand (e.g., that in Fig. 1b) and each agent's proposition set $\mathcal{P}_i$, Neary et al. [21] propose a method for deriving each agent's RM M_i by projecting M onto $\mathcal{P}_i$. We refer the reader to the paper for a description of the projection mechanism and its guarantees since we focus on learning these individual RMs instead (see Sect. 3).

Example 3. Given the local proposition sets $\mathcal{P}_1 = \{Y_B, R_B, Goal\}$, $\mathcal{P}_2 = \{Y_B, G_B, A_2^{R_B}, A_2^{-R_B}, R_B\}$ and $\mathcal{P}_3 = \{G_B, A_3^{R_B}, A_3^{-R_B}, R_B\}$, Fig. 2 shows the RMs that result from applying Neary et al.'s [21] projection algorithm.

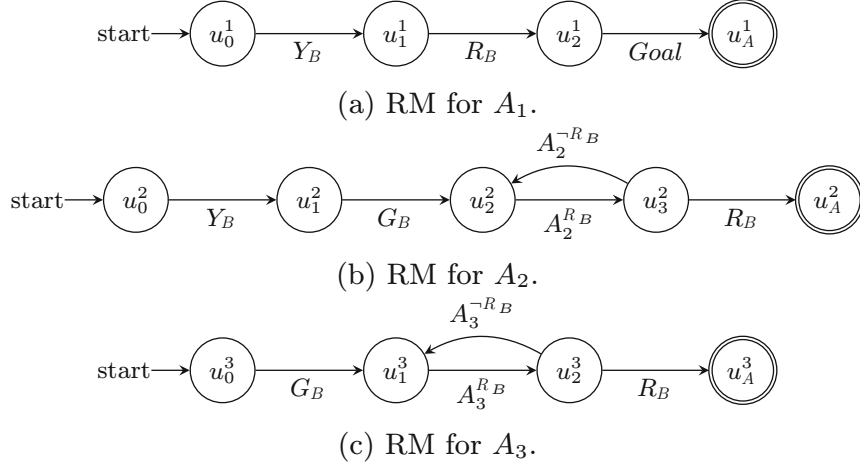


Fig. 2. RMs for each of the agents in THREEBUTTONS [21].

Neary et al. [21] extend QRM and propose a decentralised training approach to train each agent in isolation (i.e., in the absence of their teammates) exploiting their respective RMs; crucially, the learnt policies should work when all agents interact simultaneously with the world. In both the individual and team settings, agents must synchronize whenever a shared proposition is observed (i.e., a proposition that exists in the proposition set of two or more agents). Specifically, an agent should check with all its teammates whether their labelling functions also returned the same proposition. During training, given that the agent is learning in isolation and therefore the rest of the team is not actually acting, synchronization is *simulated* with a fixed probability of occurrence.

3 Learning Reward Machine in Cooperative Multi-agent Tasks

Decomposing a task using RMs is a promising approach to solving collaborative problems where coordination and synchronization among agents are required, as described in Sect. 2.2. In well-understood problems, such as the THREEBUTTONS task, one can manually engineer the RM encoding the sequence of sub-tasks needed to achieve the goal. However, this becomes a challenge for more complex problems, where the structure of the task is unknown. In fact, the human-provided RM may hinder learning if the human intuition about the structure of the task is incorrect. To alleviate this issue, we argue for learning the RM automatically from traces collected via exploration instead of handcrafting them. In what follows, we describe two approaches to accomplish this objective.

3.1 Learn a Global Reward Machine

A naive approach to learning each agent’s RM consists of two steps. First, a global RM M is learnt from traces where each label is the union of each agent’s label. Different approaches [13, 17, 29, 30] could be applied in this situation. The learnt RM is then decomposed into one per agent by projecting it onto the local proposition set $\mathcal{P}_i$ of each agent A_i [21].

Despite its simplicity, this method is prone to scale poorly. It has been observed that learning a minimal RM (i.e., an RM with the fewest states) from traces becomes significantly more difficult as the number of constituent states grows [13]. Indeed, the problem of learning a minimal finite-state machine from a set of examples is NP-complete [16]. Intuitively, the more agents interacting with the world, the larger the global RM will become, hence impacting performance and making the learning task more difficult. Moreover, having multiple agents potentially increases the number of observable propositions as well as the length of the traces; and even more so with sub-optimal policies, increasing the complexity of the problem even further.

3.2 Learn Individual Reward Machine

We propose to decompose the global task into sub-tasks, each of which can be independently solved by one of the agents. Note that even though an agent is assigned a sub-task, it does not have any information about its structure nor how to solve it. This is in fact what the proposed approach intends to tackle: learning the RM encoding the structure of the sub-task. This should be simpler than learning a global RM since the number of constituent states and propositions is smaller.

Given a set of N agents, we assume that the global task can be decomposed into N sub-task MDPs. Similarly to Neary et al. [21], we assume each agent’s MDP has its own state space $\mathcal{S}_i$, action space $\mathcal{A}_i$, transition function p_i , and termination function τ_i . In addition, each agent has its own labelling function l_i ; hence, the agent may only observe a subset of propositions $\mathcal{P}_i \subseteq \mathcal{P}$. Note that the termination function is particular to each agent; for instance, agent A_2 in `THREEBUTTONS` will complete its sub-task by pressing the green button, then the red button until it observes the signal R_B indicating that A_3 is also pressing it. Ideally, the parallel composition of the learned RMs captures the collective goal.

We propose an algorithm that *interleaves* the induction of the RMs from a collection of label traces and the learning of the associated Q-functions, akin to that by Furelos-Blanco et al. [13] for the single agent setting. The decentralised learning of the Q-functions is performed using the method by Neary et al. [21], briefly outlined in Sect. 2.2. The induction of the RMs is done using a state-of-the-art inductive logic programming system, ILASP [19], which learns the transition function of an RM as a set of logic rules from example traces observed by the agent.

Example 4. The edge connecting u_0^1 and u_1^1 in Fig. 2a corresponds to the rule $\delta(u_0^1, u_1^1, T) :- \text{prop}(Y_B, T)$. The $\delta(X, Y, T)$ atoms express that the transition from X to Y is satisfied at time T , while the $\text{prop}(P, T)$ atoms indicate that proposition P is observed at time T . The rule as a whole expresses that the transition from u_0^1 to u_1^1 is satisfied at time T if Y_B is observed at that time. The traces provided to the RM learner are expressed as sets of **prop** facts; for instance, the trace $\langle \{Y_B\}, \{R_B\}, \{Goal\} \rangle$ is mapped into $\{\text{prop}(Y_B, 0), \text{prop}(R_B, 1), \text{prop}(Goal, 2)\}$.

Algorithm 1 shows the pseudocode describing how the reinforcement and RM learning processes are interleaved. The algorithm starts initializing for each agent A_i : the set of states $\mathcal{U}^i$ of the RM, the RM M^i itself, the associated Q-functions q^i , and the sets of example goal ($\Lambda_\top^i$) and incomplete ($\Lambda_\perp^i$) traces (1.1–5). Initially, each agent’s RM solely consists of two unconnected states: the initial and final states (i.e., the agent loops in the initial state forever). Next, *NumEpisodes* are run for each agent. Before running any steps, each environment is reset to the initial state s_I^i , each RM M^i is reset to its initial state u_0^i , each agent’s trace λ^i is emptied, and we indicate that no agents have yet completed their episodes (1.8–9). Then, while there are agents that have not yet completed their episodes (1.10), a training step is performed for each of the agents, which we describe in the following paragraphs. Note that we show a sequential implementation of the approach but it could be performed in parallel to improve performance.

The **TrainAgent** routine shows how the reinforcement learning and RM learning processes are interleaved for a given agent A_i operating in its own environment. From 1.17 to 1.25, the steps for QRM (i.e., the RL steps) are performed. The agent first selects the next action a to execute in its current state s_t given the Q-function q_{u_t} associated with the current RM state u_t . The action is then applied, and the agent observes the next state s_{t+1} and whether it has achieved its goal (1.18). The next label $\mathcal{L}_{t+1}$ is then determined (1.19) and used to (i) extend the episode trace λ_t (1.20), and (ii) obtain reward r and the next RM state u_{t+1} (1.21). The Q-function q is updated for both the RM state u_t the agent is in at time t (1.22) and in a counterfactual manner for all the other states of the RM (1.25).

The learning of the RMs occurs from 1.27 to 1.42. Remember that there are two sets of example traces for each agent: one for goal traces $\Lambda_\top$ and one for incomplete traces $\Lambda_\perp$. Crucially, the learnt RMs must be such that (i) traversals for goal traces end in the final state, and (ii) traversals for incomplete traces do not end in the final state. Given the trace λ_{t+1} and the RM state u_{t+1} at time $t + 1$, the example sets are updated in three cases:

1. If λ_{t+1} is a goal trace (1.27), λ_{t+1} is added to $\Lambda_\top$. We consider as incomplete all sub-traces of $\lambda_\top$: $\{\langle \mathcal{L}_0, \dots, \mathcal{L}_k \rangle; \forall k \in \{0, \dots, |\lambda_\top| - 1\}\}$ (see Example 5). Note that if a sub-trace was a goal trace, the episode would have ended before. This optimization enables capturing incomplete traces faster; that is, it prevents waiting for them to appear as counterexamples (see next case).

Algorithm 1: Multi-agent QRM with RM Learning

```

1 for  $i = 1$  to  $N$  do
2    $\mathcal{U}^i \leftarrow \{u_0^i, u_A^i\}$ 
3    $M^i \leftarrow \text{InitializeRM}(\mathcal{U}^i)$ 
4    $q^i \leftarrow \text{InitializeQ}(M^i)$ 
5    $\Lambda_\top^i \leftarrow \emptyset, \Lambda_\perp^i \leftarrow \emptyset$ 
6 for  $n = 1$  to  $\text{NumEpisodes}$  do
7    $t \leftarrow 0$ 
8   for  $i = 1$  to  $N$  do
9      $u_t^i \leftarrow u_0^i, s_t^i \leftarrow s_I^i, \lambda_t^i \leftarrow \emptyset, \text{done}^i \leftarrow \perp$ 
10  while  $\exists j \in N$  such that  $\text{done}^j = \perp$  do
11    for  $i = 1$  to  $N$  do
12       $\text{TrainAgent}(\text{done}^i, s_t^i, u_t^i, q^i, \lambda_t^i, \Lambda_G^i, \Lambda_I^i, M^i, \mathcal{U}^i)$ 
13       $t \leftarrow t + 1$ 
14
15 Procedure  $\text{TrainAgent}(\text{done}, s_t, u_t, q, \lambda_t, \Lambda_\top, \Lambda_\perp, M, \mathcal{U})$ 
16   if  $\text{done} = \perp$  then
17      $a \leftarrow \text{GetAction}(q_{u_t}, s_t)$ 
18      $s_{t+1}, \text{isGoalAchieved} \leftarrow \text{EnvStep}(s_t, a)$ 
19      $\mathcal{L}_{t+1} \leftarrow l(s_t, s_{t+1})$ 
20      $\lambda_{t+1} \leftarrow \lambda_t \oplus \mathcal{L}_{t+1}$ 
21      $r, u_{t+1} \leftarrow \text{GetNextRMState}(M, u_t, \mathcal{L}_{t+1})$ 
22      $q_{u_t} \leftarrow \text{UpdateQ}(q_{u_t}, s_t, a, r, s_{t+1}, q_{u_{t+1}})$ 
23     for  $u \in \mathcal{U} \setminus \{u_t\}$  do
24        $r, u' \leftarrow \text{GetNextRMState}(M, u, \mathcal{L}_{t+1})$ 
25        $q_u \leftarrow \text{UpdateQ}(q_u, s_t, a, r, s_{t+1}, q_{u'})$ 
26
27     if  $\text{isGoalAchieved}$  then
28        $\Lambda_\top \leftarrow \Lambda_\top \cup \{\lambda_{t+1}\}$ 
29        $\Lambda_\perp \leftarrow \Lambda_\perp \cup \text{GenerateIncompleteTraces}(\lambda_{t+1})$ 
30        $M_{\text{new}} \leftarrow \text{LearnRM}(\mathcal{U}, \Lambda_\top, \Lambda_\perp)$ 
31        $\text{done} \leftarrow \top$ 
32     else if  $u_{t+1} = u_A$  then
33        $\Lambda_\perp \leftarrow \Lambda_\perp \cup \{\lambda_{t+1}\}$ 
34        $M_{\text{new}} \leftarrow \text{LearnRM}(\mathcal{U}, \Lambda_\top, \Lambda_\perp)$ 
35        $\text{done} \leftarrow \top$ 
36     else if  $t + 1 = T$  then
37        $\Lambda_\perp \leftarrow \Lambda_\perp \cup \{\lambda_{t+1}\}$ 
38        $\text{done} \leftarrow \top$ 
39
40     if  $M \neq M_{\text{new}}$  then
41        $M \leftarrow M_{\text{new}}$ 
42        $q \leftarrow \text{InitializeQ}(M)$ 
43
44 Procedure  $\text{LearnRM}(\mathcal{U}, \Lambda_\top, \Lambda_\perp)$ 
45    $M \leftarrow \text{RMLeaer}(\mathcal{U}, \Lambda_\top, \Lambda_\perp)$ 
46   while  $M = \perp$  do
47      $\mathcal{U} \leftarrow \mathcal{U} \cup \{u_{|\mathcal{U}|-1}\}$ 
48      $M \leftarrow \text{RMLeaer}(\mathcal{U}, \Lambda_\top, \Lambda_\perp)$ 
49   return  $M$ 

```

2. If λ_{t+1} is an incomplete trace and u_{t+1} is the final state of the RM (1.32), then λ_{t+1} is a counterexample since reaching the final state of the RM should be associated with completing the task. In this case, we add λ_{t+1} to $\Lambda_{\perp}$.
3. If λ_{t+1} is an incomplete trace and the maximum number of steps per episode T has been reached (1.36), we add λ_{t+1} to $\Lambda_{\perp}$.

In the first two cases, a new RM is learnt once the example sets have been updated. The **LearnRM** routine (1.44–49) finds the RM with the fewest states that covers the example traces. The RM learner (here ILASP) is initially called using the current set of RM states $\mathcal{U}$ (1.45); then, if the RM learner finds no solution covering the provided set of examples, the number of states is increased by 1. This approach guarantees that the RMs learnt at the end of the process are *minimal* (i.e., consist of the fewest possible states) [13]. Finally, the Q-functions associated with an RM are reset when a new RM is learnt; importantly, the Q-functions depend on the RM structure and, hence, they are not easy to keep when the RM changes (1.42). In all three enumerated cases, the episode is interrupted.

Example 5. Given the sub-task assigned to the agent A_3 in the **THREEBUTTONS** environment described in Fig. 2c, we consider a valid goal trace $\lambda = \langle \{\}, \{G_B\}, \{A_3^{R_B}\}, \{A_3^{-R_B}\}, \{A_3^{R_B}\}, \{R_B\} \rangle$. The generated incomplete traces are:

$$\begin{aligned}
 &\langle \{\} \rangle, \\
 &\langle \{\}, \{G_B\} \rangle, \\
 &\langle \{\}, \{G_B\}, \{A_3^{R_B}\} \rangle, \\
 &\langle \{\}, \{G_B\}, \{A_3^{R_B}\}, \{A_3^{-R_B}\} \rangle, \\
 &\langle \{\}, \{G_B\}, \{A_3^{R_B}\}, \{A_3^{-R_B}\}, \{A_3^{R_B}\} \rangle.
 \end{aligned}$$

4 Experiments

We evaluate our approach by looking at two metrics as the training progresses: (1) the collective reward obtained by the team of agents and (2) the number of steps required for the team to achieve the goal. The first metric indicates whether the team learns how to solve the task by getting a reward of 1 upon completion. The second one estimates the quality of the solution, i.e. the lower the number of steps the more efficient the agents. Although the training is performed in isolation for each agent, the results presented here were evaluated with all the agents acting together in a shared environment. Additionally, we perform a qualitative assessment of the RM learnt by each agent.

For all the experiments, we show in *green* the performance for the case where RMs are known a priori and provided to the agents. This scenario, where we have perfect knowledge about the structure of each sub-task, is used as an upper bound for the results of our approach. In *pink*, we show the performances of our approach where the local RMs are learnt. The *yellow* curve shows the performance when the agents learn to perform their individual task without the RM

construct.¹ The results presented show the mean value and the 95%-confidence interval of the metrics evaluated over 5 different random seeds in two grid-based environments of size 7×7 . We ran the experiments on an EC2 C5.2XLARGE instance on AWS using *Python* 3.7.0. We use the state-of-the-art inductive logic programming system *ILASP* v4.2.0 [19] to learn RMs from traces with a 1h timeout.

4.1 THREEBUTTONS Task

We evaluate our approach in the THREEBUTTONS task described in Example 1. The number of steps required to complete the task and the collective reward obtained by the team of agents throughout training are shown in Fig. 3. We observe that regardless of whether the local RMs are handcrafted or learnt, the agents learn policies that, when combined, lead to the completion of the global task. Indeed, the obtained collective reward converges to the maximum achievable indicating that all the agents have learnt to execute their sub-tasks and coordinate to achieve the common goal. Learning the RMs also helps speed up the learning for each of the agents, significantly reducing the number of training iterations required to learn to achieve the global task. However, we note that the policy learnt without the RM framework requires fewer steps on average than when the RMs are used.

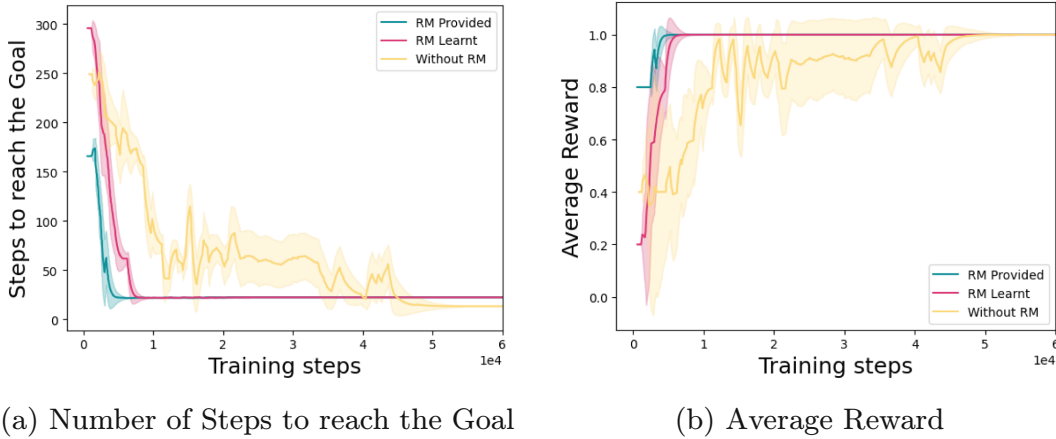


Fig. 3. Comparison between handcrafted RMs (RM Provided) and our approach learning the RMs from traces (RM Learnt) in the THREEBUTTONS environment.

We present in Fig. 4 the RM learnt by A_2 using our interleaved learning approach. We compare it to the “true” RM shown in Fig. 2b. Although important to truly reflect the dynamics of the environment and the labelling function, the back transition from u_3^2 to u_2^2 associated with the label A_2^{RB} is missing

¹ This is equivalent of using a 2-state RM, reaching the final state only when the task is completed.

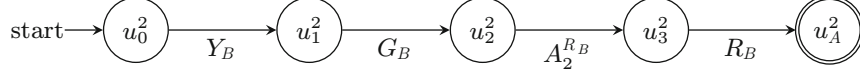


Fig. 4. Learnt RM for A_2 .

from the learnt RM. In fact, A_2 must not only press the red button ($A_2^{R_B}$) but also remain on it until the event R_B is observed, or else R_B will never occur. The missing transition can be explained by the fact that the RM is learnt from *positive* (i.e., observable) examples only; that is, there are no *negative* (i.e., unfeasible) examples where the event R_B is observed directly after $A_2^{R_B}$. This additional knowledge would have helped the RM learner identify this trace as impossible and realise that R_B can only be observed if A_2 does not stop pressing the red button ($A_2^{R_B}$). However, the joint learning of the RM and the associated optimal RL policies mitigates this problem. Due to the discounting factor and the fact that a non-zero reward is only given upon completing the task, the RL policies learn to minimize the number of steps required to achieve the task; hence, optimal policies will never consider travelling backwards in the RM. The RL policy associated with u_2^2 thus converges towards having A_2 staying on the red button until R_B is observed, removing the need to consider the back transition from u_3^2 to u_2^2 .

Learning the global RM was also attempted in this environment. Unfortunately, learning the minimal RM for the size of this problem is hard and the RM learner timed out after the $1h$ limit.

4.2 RENDEZVOUS Task

The second environment in which we evaluate our approach is the 2-agent RENDEZVOUS task [21] presented in Fig. 5a. This task is composed of 2 agents acting in a grid with the ability to go UP, DOWN, LEFT, RIGHT or DO NOTHING. The task requires the agents to move in the environment to first meet in an *RDV*

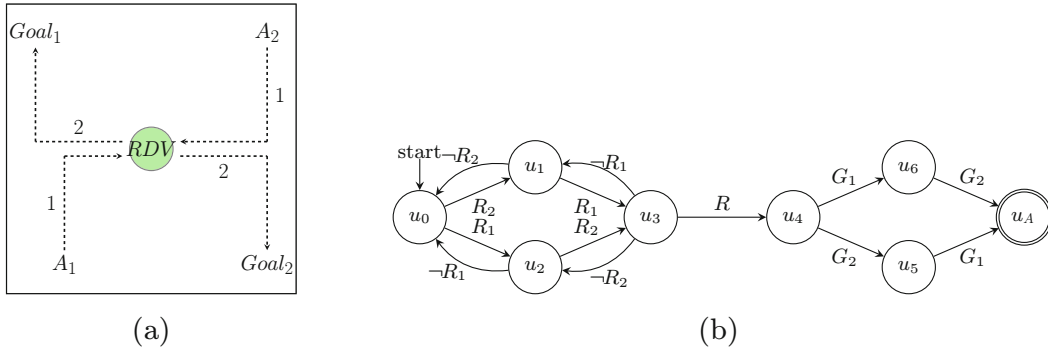


Fig. 5. Example of the RENDEZVOUS task where 2 agents must meet on the RDV point (green) before reaching their goal state G_1 and G_2 for agents A_1 and A_2 respectively. (Color figure online)

location, i.e. all agents need to simultaneously be in that location for at least one timestep. Then each of the agents must reach their individual goal location for the global task to be completed. The RM describing the task is shown in Fig. 5b.

We present in Fig. 6 the number of steps (6a) to complete the task and the collective reward received by the team of agents (6b) throughout training. The collective reward obtained while interleaving the learning of the policies and the learning of the local RMs converges to the maximum reward after a few iterations. In this task, the RM framework is shown to be crucial to helping the team solve the task in a timely manner. Indeed, without the RMs (either known a priori or learnt), the team of agents do not succeed at solving the task.

We also attempted to learn the global RM directly but the task could not be solved (i.e., a collective goal trace was not observed). The collective reward remained null throughout the training that we manually stopped after 1×10^6 timesteps.

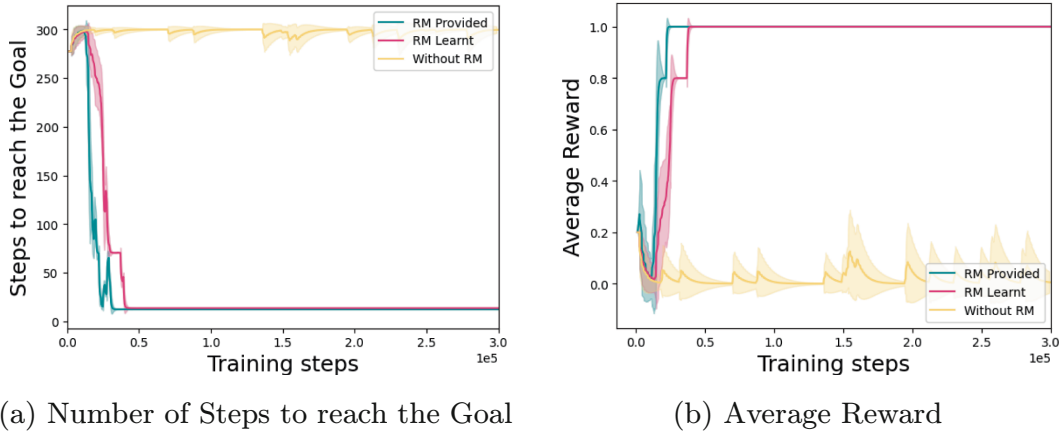


Fig. 6. Comparison between handcrafted RMs (RM Provided) and our approach learning the RMs from traces (RM Learnt) in the RENDEZVOUS environment.

5 Related Work

Since the recent introduction of RMs as a way to model non-Markovian rewards [27, 28], several lines of research have based their work on this concept (or similar finite-state machines). Most of the work has focused on how to derive them from logic specifications [4] or demonstrations [5], as well as learning them using different methods [6, 13–15, 17, 29, 30].

In this work, based on the multi-agent with RMs work by Neary et al. [21], the RMs of the different agents are executed in parallel. Recent works have considered other ways of composing RMs, such as merging the state and reward transition functions of two RMs [9] or arranging them hierarchically by enabling RMs to call each other [14]. The latter is a natural way of extending this work since agents could share subroutines in the form of callable RMs.

The policy learning algorithm we employ for exploiting RMs is an extension of the QRM algorithm (see Sect. 2.2). However, algorithms based on making decisions at two hierarchical levels [13, 27] or more [14] have been proposed. In the simplest case (i.e., two levels), the agent first decides which transition to satisfy from a given RM state, and then decides which low-level actions to take to satisfy that transition. For example, if agent A_2 gets to state u_3^2 in Fig. 2b, the decisions are: (1) whether to satisfy the transition labelled $A_2^{-R_B}$ or the one labelled R_B , and (2) given the chosen transition, act towards satisfying it. Unlike QRM, this method is not guaranteed to learn optimal policies; however, it enables lower-level sub-tasks to be reused while QRM does not. As the Q-functions learned by QRM depend on the structure of the whole RM, they must be re-initialized every time a new RM is learnt. While other methods attempt to leverage the previously learned Q-functions [30], the hierarchical approach outlined here does not need to re-learn all functions each time a new RM is induced.

In the multi-agent community, the use of finite-state machines for task decomposition in collaborative settings has been studied in [7, 11] where a top-down approach is adopted to construct sub-machines from a global task. Unlike our work, these methods focus on the decomposition of the task but not on how to execute it. The algorithm presented in this paper interleaves both the decomposition and the learning of the policies associated with each state of the RM.

The MARL community studying the collaborative setting has proposed different approaches to decompose the task among all the different agents. For example, QMIX [22] tackles the credit assignment problem in MARL assuming that the team’s Q-function can be factorized, and performs a value-iteration method to centrally train policies that can be executed in a decentralised fashion. While this approach has shown impressive empirical results, its interpretation is more difficult than understanding the structure of the task using RM.

The use of RMs in the MARL literature is starting to emerge. The work of Neary et al. [21] was the first to propose the use of RMs for task decomposition among multiple agents. As already mentioned in this paper, this work assumes that the structure of the task is known ‘a priori’, which is often untrue. With a different purpose, Dann et al. [8] propose to use RMs to help an agent predict the next actions the other agents in the system will perform. Instead of modelling the other agents’ plan (or program), the authors argue that RMs provide a more permissive way to accommodate for variations in the behaviour of the other agent by providing a higher-level mechanism to specify the structure of the task. In this work as well, the RMs are pre-defined and provided to the agent.

Finally, the work of Eappen and Jagannathan [10] uses a finite-state machine called ‘task monitor’, which is built from temporal logic specifications that encode the reward signal of the agents in the system. A task monitor is akin to an RM with some subtle differences like the use of registers for memory.

6 Conclusion and Future Work

Dividing a problem into smaller parts that are easier to solve is a natural approach performed instinctively by humans. It is also particularly suited for the

multi-agent setting where all the available resources can be utilized to solve the problem or when a complementary set of skills is required.

In this work, we extend the approach by Neary et al. [21], who leverage RMs for task decomposition in a multi-agent setting. We propose a novel approach where the RMs are learnt instead of manually engineered. The method presented in this paper interleaves the learning of the RM from the traces collected by the agent and the learning of the set of policies used to achieve the task’s goal. Learning the RMs associated with each sub-task not only helps deal with non-Markovian rewards but also provides a more interpretable way to understand the structure of the task. We show experimentally in the `THREEBUTTONS` and the `RENDEZVOUS` environments that our approach converges to the maximum reward a team of agents can obtain by collaborating to achieve a goal. Finally, we validated that the learnt RM corresponds indeed to the RM needed to achieve each sub-task.

While our approach lifts the assumption about knowing the structure of the task ‘a priori’, a set of challenges remain and could be the object of further work:

1. A labelling function is assumed to exist for each of the agents and be known by them. In other words, each agent knows the labels relevant to its task and any labels used for synchronization with the other team members.
2. Noisy labelling functions are currently not supported by our approach but could be the object of future work. Extending the RM framework with probabilistic transition would be required adding another level of complexity to the RM learning process.
3. Our approach assumes that the global task has been automatically divided into several tasks, each to be completed by one of the agents. Dynamically creating these sub-tasks and assigning them to each of the agents is crucial to having more autonomous agents and would make an interesting topic for further research.
4. We leverage task decomposition to learn simpler RMs that when run in parallel represent a more complex global RM. Task decomposition can be driven further through task hierarchies, which could be represented through hierarchies of RMs [14]. Intuitively, each RM in the hierarchy should be simpler to learn and enable reusability and further parallelization among the different agents.
5. Finally, to allow even more reusability among the different agents and learn “skills” instead of tasks. The framework could be extended to learn transition using first-order formulae rather than propositions. This could potentially reduce the number of RL policies to learn and allow generalization of the skills learnt.

The use of RMs in the multi-agent context is still in its infancy and there are several exciting avenues for future work. We have demonstrated in this paper that it is possible to learn them under certain assumptions that could be relaxed in future work.

Acknowledgements. Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-22-2-0243. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorised to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

1. Albrecht, S.V., Christianos, F., Schäfer, L.: Multi-Agent Reinforcement Learning: Foundations and Modern Approaches. MIT Press, Cambridge (2023)
2. Ardon, L., Vadori, N., Spooner, T., Xu, M., Vann, J., Ganesh, S.: Towards a fully RL-based market simulator. In: Proceedings of the ACM International Conference on AI in Finance (ICAIF), pp. 7:1–7:9 (2021)
3. Busoniu, L., Babuska, R., De Schutter, B.: A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.)* **38**(2), 156–172 (2008)
4. Camacho, A., Toro Icarte, R., Klassen, T.Q., Valenzano, R.A., McIlraith, S.A.: LTL and beyond: formal languages for reward function specification in reinforcement learning. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 6065–6073 (2019)
5. Camacho, A., Varley, J., Zeng, A., Jain, D., Iscen, A., Kalashnikov, D.: Reward machines for vision-based robotic manipulation. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 14284–14290 (2021)
6. Christoffersen, P.J.K., Li, A.C., Toro Icarte, R., McIlraith, S.A.: Learning symbolic representations for reinforcement learning of non-Markovian behavior. In: Proceedings of the Knowledge Representation and Reasoning Meets Machine Learning (KR2ML) Workshop at the Advances in Neural Information Processing Systems (NeurIPS) Conference (2020)
7. Dai, J., Lin, H.: Automatic synthesis of cooperative multi-agent systems. In: Proceedings of the IEEE Conference on Decision and Control (CDC), pp. 6173–6178 (2014)
8. Dann, M., Yao, Y., Alechina, N., Logan, B., Thangarajah, J.: Multi-agent intention progression with reward machines. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 215–222 (2022)
9. De Giacomo, G., Favorito, M., Iocchi, L., Patrizi, F., Ronca, A.: Temporal logic monitoring rewards via transducers. In: Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR), pp. 860–870 (2020)
10. Eappen, J., Jagannathan, S.: DistSPECTRL: distributing specifications in multi-agent reinforcement learning systems. arXiv preprint [arXiv:2206.13754](https://arxiv.org/abs/2206.13754) (2022)
11. Elsefy, A.E.: A task decomposition using (HDec-POSMDPs) approach for multi-robot exploration and fire searching. *Int. J. Robot. Mechatron.* **7**(1), 22–30 (2020)
12. Fuchs, F., Song, Y., Kaufmann, E., Scaramuzza, D., Durr, P.: Super-human performance in gran turismo sport using deep reinforcement learning. *IEEE Robot. Autom. Lett.* **6**(3), 4257–4264 (2021)
13. Furelos-Blanco, D., Law, M., Jonsson, A., Broda, K., Russo, A.: Induction and exploitation of subgoal automata for reinforcement learning. *J. Artif. Intell. Res.* **70**, 1031–1116 (2021)

14. Furelos-Blanco, D., Law, M., Jonsson, A., Broda, K., Russo, A.: Hierarchies of reward machines. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 10494–10541 (2023)
15. Gaon, M., Brafman, R.I.: Reinforcement learning with non-Markovian rewards. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pp. 3980–3987 (2020)
16. Gold, E.M.: Complexity of automaton identification from given data. *Inf. Control* **37**(3), 302–320 (1978)
17. Hasanbeig, M., Jeppu, N.Y., Abate, A., Melham, T., Kroening, D.: DeepSynth: automata synthesis for automatic task segmentation in deep reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pp. 7647–7656 (2021)
18. Kaelbling, L.P.: Learning to achieve goals. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 1094–1099 (1993)
19. Law, M., Russo, A., Broda, K.: The ILASP System for Learning Answer Set Programs (2015). www.ilasp.com
20. Mnih, V., et al.: Human-level control through deep reinforcement learning. *Nature* **518**(7540), 529–533 (2015)
21. Neary, C., Xu, Z., Wu, B., Topcu, U.: Reward machines for cooperative multi-agent reinforcement learning. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 934–942 (2021)
22. Rashid, T., Samvelyan, M., De Witt, C.S., Farquhar, G., Foerster, J., Whiteson, S.: Monotonic value function factorisation for deep multi-agent reinforcement learning. *J. Mach. Learn. Res.* **21**(1), 7234–7284 (2020)
23. Shalev-Shwartz, S., Shammah, S., Shashua, A.: Safe, multi-agent, reinforcement learning for autonomous driving. arXiv preprint [arXiv:1610.03295](https://arxiv.org/abs/1610.03295) (2016)
24. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. *Nature* **529**(7587), 484–489 (2016)
25. Sultana, N.N., Meisheri, H., Baniwal, V., Nath, S., Ravindran, B., Khadilkar, H.: Reinforcement learning for multi-product multi-node inventory management in supply chains. arXiv preprint [arXiv:2006.04037](https://arxiv.org/abs/2006.04037) (2020)
26. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (2018)
27. Toro Icarte, R., Klassen, T., Valenzano, R., McIlraith, S.: Using reward machines for high-level task specification and decomposition in reinforcement learning. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 2107–2116 (2018)
28. Toro Icarte, R., Klassen, T.Q., Valenzano, R., McIlraith, S.A.: Reward machines: exploiting reward function structure in reinforcement learning. *J. Artif. Intell. Res.* **73**, 173–208 (2022)
29. Toro Icarte, R., Waldie, E., Klassen, T.Q., Valenzano, R.A., Castro, M.P., McIlraith, S.A.: Learning reward machines for partially observable reinforcement learning. In: Proceedings of the Advances in Neural Information Processing Systems (NeurIPS) Conference, pp. 15497–15508 (2019)
30. Xu, Z., et al.: Joint inference of reward machines and policies for reinforcement learning. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), pp. 590–598 (2020)

Author Queries

Chapter 3

Query Refs.	Details Required	Author's response
AQ1	As Per Springer style, both city and country names must be present in the affiliations. Accordingly, we have inserted the city name "London" in the affiliation. Please check and confirm if the inserted city name is correct. If not, please provide us with the correct city name.	That is OK.