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

Automatic Decomposition of Reward Machines for Decentralized Multiagent Reinforcement Learning

Sophia Smith, Cyrus Neary, and Ufuk Topcu

Abstract— **In cooperative multiagent reinforcement learning (MARL), a team of agents learns to work together to complete a task. Centralized approaches to MARL quickly become intractable as the number of agents increases, necessitating decentralized learning algorithms which take advantage of task decompositions to train the agents individually. However, these task decompositions typically require careful human engineering. In this work, we develop algorithms to automatically decompose a team task into a collection of subtasks that can be used for decentralized reinforcement learning. We use reward machines—structured representations of reward functions—to encode team tasks and to automatically generate task decompositions that enforce the following three properties. 1) *Task Consistency*: We generate decompositions that are consistent with the team's task—if the agents individually learn to accomplish their subtasks, we guarantee that the composition of their learned behaviors will accomplish the original task. 2) *Minimized Coordination*: Inter-agent coordination during task execution can be costly. We minimize the coordination that's necessary to execute the decomposed tasks, which simplifies the decentralized learning problem by reducing each agent's interdependencies with its teammates. 3) *Fairly Distributed*: We maximize a weighted sum that balances the total utility of the agents and the *fairness* of the decomposition, which we define in terms of the distribution of assigned subtasks between the agents. Experimental results in three-agent and five-agent MARL tasks show the method's novel capabilities. The algorithm automatically generates task decompositions that are consistent with the team task, that reduce unnecessary coordination between the agents, and that take the agent's utility over subtasks into account. When used to define decentralized objectives, the generated task decompositions result in team policies that efficiently complete the task. Meanwhile, baseline decompositions yield policies that fail to complete the task.**

I. Introduction

In cooperative multiagent reinforcement learning (MARL), a team of agents learns to complete a common objective through interactions with each other and with a shared task environment. The computational and data requirements of centralized approaches to MARL, in which the single decision-making policy is trained using data collected from the entire team, scale exponentially in the number of agents. These issues necessitate decentralized MARL algorithms, which learn separate policies for the individual agents in a distributed fashion.

One approach to decentralized RL is to decompose the team task into subtasks to be completed by individual agents. The agents can then be trained individually on their assigned subtasks, simplifying the learning problem. However, in general such task decompositions must be constructed manually, and they can be difficult to design in practice.

In this work, we present a framework and algorithms for the automatic decomposition of cooperative MARL tasks. Specifically, we use reward machines (RM)—finite transitions systems that encode reinforcement learning objectives—to represent the team tasks. This structured representation enables formal analysis and computation of task decompositions; the task RM can be decomposed into a collection of *subtask* RMs, each of which encodes the required local behavior of an individual agent [1].

Using the team task RM, we first define a *strategy reward machine*—a simpler version of the RM that only encodes one of the (potentially many) ways that the team could complete the task. We then present four conditions that we use to define *feasible* decompositions of this strategy RM. We use these conditions to check whether a candidate decomposition will result in decentralized behavior that both completes the team task and is consistent with the heterogeneous capabilities of the individual agents. Furthermore, we define score functions to compare the feasible decompositions in terms of their relative fairness, the total utility they provide to the agents, and the amount of coordination they require from the agents at runtime. Finally, we present an algorithm to automatically search for a decomposition that satisfies the aforementioned conditions and that maximizes a weighted combination of these score functions.

We apply decentralized algorithms for MARL to train the agents to complete the automatically generated task decompositions. To train the individual agents to complete their local subtasks while simultaneously preventing them from interfering with the local subtasks of their teammates, we introduce the notion of an *accident avoidance* RM, which we use to define the reward signals observed by the agents during training.

We demonstrate our proposed task decomposition algorithm through three-agent and five-agent decentralized MARL tasks. The agents efficiently learn to complete the team tasks using the automatically generated decompositions. Meanwhile, decentralized MARL using a baseline decomposition algorithm fails to learn successful behavior.

II. Related Work

Task decomposition of multiagent systems with known system models has been approached in several ways. For

This work was supported by the Army Research Office (ARO W911NF-20-1-0140) and the Office of Naval Research (ONR N00014-20-1-2115).

S. Smith and C. Neary are with the Oden Institute for Computational Engineering and Sciences, while U. Topcu is with the Oden Institute as well as the Department of Aerospace Engineering and Engineering Mechanics at the University of Texas at Austin, Austin, TX, USA. Email: {smith.sj, cneary, utopcu}@utexas.edu.

example, [2] and [3] study conditions for the decomposition of team task automata and [2] presents a hierarchical algorithm for generating task decompositions. [4] and [5] take an automata-based approach to identify potential decompositions of an LTL task specification. Similar to our work, [6] study methods to minimize the necessary coordination between agents during the execution of a team task. However, these approaches assume the existence of a known environment model, while we consider task decompositions to be used in a learning setting.

Meanwhile, [7], [8] use reward machines, finite-state machines encoding temporally extended tasks in terms of atomic propositions, to break tasks into stages for which separate policies can be learned. [9] and [10] propose to simultaneously learn reward machines and RL policies. Similarly, [11] propose to learn automata-based representations of RL tasks. However, these works apply reward machines to single agent RL problems, whereas our primary focus is on their use for task decomposition in the MARL setting.

Algorithms to solve MARL problems have been studied extensively [12], [13], [14], [15]. Many of these algorithms, such as independent q-learning [16], offer decentralized approaches to training: the agents independently apply reinforcement learning algorithms on data that is generated by their collective interactions. Other so-called *centralized training decentralized execution* methods instead update the agents' policies in a centralized fashion, while ensuring that the learned policies may be executed in a decentralized fashion at runtime [17], [18], [19]. These works, similar to [20], [21], typically decompose the joint q-function itself to allow for decentralized execution. However, we note that all of the aforementioned works train the agents simultaneously and thus either suffer from the multiagent coordination problem [22], the non-stationarity of the learning problem [13], or from the sample inefficiency that arises during centralized training. By contrast, we use reward machines as a method to specify subtasks and to train individual agents to complete them, in the absence of their teammates.

A number of works have recently applied techniques and ideas from the formal methods community to MARL problems. The authors of [23] present extended Markov games; a mathematical model allowing multiple agents to concurrently learn to satisfy multiple non-Markovian task specifications. [24], [25] present automata and logic-based reward shaping methods for multiagent reinforcement learning. Similarly, [26] present a method for logic-based multiagent reward shaping while enabling distributed training for the local objectives of the agents.

The authors of [1] use reward machines to specify team task decompositions and to design decentralized MARL algorithms. We build on this work by developing techniques to synthesize such task decompositions automatically, while ensuring that the criteria necessary for correct task decomposition are met. This relaxes the assumption of user-specified knowledge of the task decomposition. Furthermore, we relax the conditions presented in [1] for a reward machine to be used in decentralized MARL, and we present methods to quantitatively compare task decompositions. These contributions are necessary for the automation of compositional approaches to decentralized MARL.

III. Preliminaries

A. Markov Decision Processes and Stochastic Games

In single agent reinforcement learning (RL), we use Markov Decision Processes (MDPs) to model the task environment. An MDP is a tuple $\mathcal{M} = \langle S, A, p, r, \gamma \rangle$, with a finite set of states S, a finite set of actions A, a transition probability function $p : S \times A \to \Delta(S)$ where $\Delta(S)$ is the set of all probability distributions over S, a reward function $r : S \times A \times S \to \mathbb{R}$ and a discount factor $\gamma \in (0, 1]$. A stationary policy $\pi : S \to \Delta(A)$ maps a state $s \in S$ to a probability distribution over actions. The objective in RL is to find an optimal policy π^* that maximizes the expected discounted sum of future rewards, from any state.

We denote a team of n agents by $\Lambda = (\alpha_1, \ldots, \alpha_n)$. In multiagent reinforcement learning (MARL), the environment is instead modeled by a stochastic game—the multiagent extension of an MDP.

Definition 1 (Stochastic Games)**.** *A stochastic game is a tuple $\mathcal{G} = \langle S_1, \ldots, S_n, A_1, \ldots, A_n, p, R, \gamma \rangle$. In a stochastic game, S_i is the finite set of states of agent i, and A_i the finite set of actions available to agent i. We define the joint states of the team of n agents as $\mathcal{S} = S_1 \times S_2 \ldots \times S_n$, the joint set of actions as $\mathcal{A} = A_1 \times A_2 \ldots \times A_n$, the transition function as $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ (where $\Delta(\mathcal{S})$ is a probability distribution over $\mathcal{S}$), the reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, and the discount factor $\gamma \in (0, 1]$.*

We assume local transition probability functions $p_i : S_i \times A_i \to \Delta(S_i)$ independently govern the dynamics of each agent. From the local transition probability functions, the joint transition function is constructed as $p(\mathbf{s}'|\mathbf{s}, \mathbf{a}) = \Pi_{i=1}^{n} p_i(s_i'|s_i, a_i)$ for all joint states $\mathbf{s}$, $\mathbf{s}' \in \mathcal{S}$ and joint actions $\mathbf{a} \in \mathcal{A}$. We note that although the stochastic state transitions of the agents are independent, their optimal policies may still depend on their teammates' states as a result of the joint reward function.

In a stochastic game, a team policy is defined as $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. Similar to RL, the objective in MARL is to find a policy π^* that maximizes the expected discounted reward from any joint state $s \in S$. In this work, we consider team policies that are defined in terms of the *local policies* $\pi_i : S_i \to \Delta(A_i)$ of the individual agents. That is, we define $\pi(\mathbf{a}|\mathbf{s}) = \prod_{i=1}^{n} \pi_i(a_i|s_i)$, where $s_i \in S_i$ and $a_i \in A_i$ are the i^{th} components of the joint states $\mathbf{s} \in S$ and joint actions $\mathbf{a} \in A$, respectively.

B. Reward Machines

Definition 2 (Reward Machine)**.** *A reward machine is defined by the tuple, $\mathcal{R} = \langle U, \Sigma, u_I, F, \delta, \sigma \rangle$ with states U, events Σ, initial state u_I, terminal states $F \subset U$, transition function $\delta : U \times \Sigma \to U$, and reward function $\sigma : U \times U \to \mathbb{R}$.*

Intuitively, we use reward machines to capture the steps required for an agent (or team of agents) to complete their

task. Each state represents a separate stage of the task. For example, the reward machine shown in Fig. 1 encodes the following task: two agents work together to chop down a tree, the tree falls, one agent transports the log to the work bench, and finally the experienced woodworker carves something from the log, earning the team a reward. Here, each ai event corresponds to agent i arriving at the tree to help cut down the tree and li corresponds to agent i leaving the tree before it falls down. The event $timber$ signals the tree falling, each tri corresponds to agent i transporting the log to the craft table, ar signals the successful arrival of the log, and $craft$ represents the woodworker successfully crafting the log.

In this work, we limit our discussion to *task completion* reward machines: the agents only receive a reward if they reach a final state. The reward function is defined as $\sigma(u_1, u_2) = 1$ if $u_1 \notin F$ and $u_2 \in F$, and $\sigma(u_1, u_2) = 0$ otherwise.

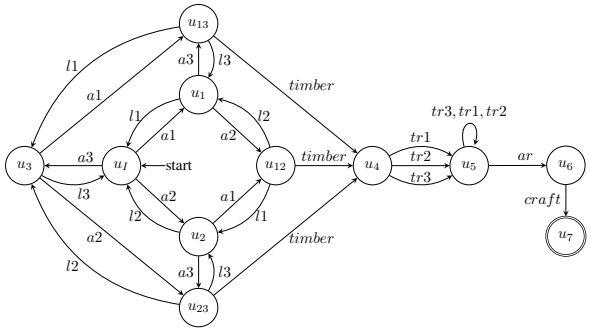
We note that functions δ and σ are partial functions, only defined for subsets of U and Σ. A run on a sequence of events $\xi = e_0...e_k \in \Sigma^*$ is the sequence $u_0e_0u_1e_1...u_ke_ku_{k+1}$, where $u_i \in U$ for $i = 0, 1, ...k+1$, $u_0 = u_I$ and $\delta(u_i, e_i) = u_{i+1}$ for all $i = 0, 1, ..., k$. Since δ and σ are partial functions, not every sequence of events $\xi \in \Sigma^*$ has a valid run on a reward machine $\mathcal{R}$. We define $Tr(\mathcal{R})$ as the set of all event sequences that have valid runs on $\mathcal{R}$. For an event sequence $\xi \in Tr(\mathcal{R})$, $\mathcal{R}(\xi)$ is the reward output of the corresponding run. Specifically, for $\xi = e_0...e_k$, $\mathcal{R}(\xi) = 1$ if $u_{k+1} \in F$ and $u_k \notin F$ and $\mathcal{R}(\xi) = 0$ otherwise. Additionally we define $reach(u_0, \mathcal{R})|_{\Sigma_0} \subset U$, where $u \in reach(u_0, \mathcal{R})|_{\Sigma_0}$ if there exists a transition sequence in $\mathcal{R}$ starting from u_0 and ending at u using only events in Σ_0.

We define reward machine projections following [1]. Intuitively, a projection of the team reward machine onto an agent's *local event set* $\Sigma_i \subset \Sigma$ represents the team task from the point of view of agent i, who may only observe events from Σ_i. The projection of a reward machine onto an event space $\Sigma_i \subset \Sigma$ is formally defined as the tuple $\text{Proj}(\mathcal{R})|_{\Sigma_i} = \mathcal{R}_i = \langle U_i, u_I^i, \Sigma_i, \delta_i, \sigma_i, F_i \rangle$ where $U_i = U/\sim_i$ is a set of equivalence classes over U for the equivalence relation $\sim_i$ defined in [1], u_I^i is the equivalence class containing u_I, F_i is the equivalence class containing F, $\delta_i : U_i \times \Sigma_i \to U_i$ is the projected transition function defined such that $u_2^i = \delta_i(u_1^i, e)$ if and only if there exist $u_1, u_2 \in U$ such that $u_1^i = [u_1]_i$, $u_2^i = [u_2]_i$, and $u_2 = \delta(u_1, e)$, and $\sigma_i : U_i \times U_i \to \mathbb{R}$ is the projected output function defined such that $\sigma_i(u_1^i, u_2^i) = 1$ if $u_1^i \notin F_i, u_2^i \in F_i$ and $\sigma(u_1^i, u_2^i) = 0$ otherwise. In our definition of δ_i, $[u_1]_i \in U_i$ denotes the equivalence class, defined by equivalence relation $\sim_i$, that contains $u_1 \in U$.

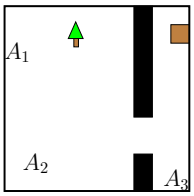
Given an event sequence $\xi \in \Sigma^*$, we define the projection of the sequence onto local event set Σ_i, $P_{\Sigma_i}(\xi) \in \Sigma_i^*$, recursively using the relationships $P_{\Sigma_i}(\varepsilon) = \varepsilon$, $P_{\Sigma_i}(\xi e) = P_{\Sigma_i}(\xi)e$ if $e \in \Sigma_i$, and $P_{\Sigma_i}(\xi e) = P_{\Sigma_i}(\xi)$ if $e \notin \Sigma_i$ for any $\xi \in \Sigma^*$. Here, ε denotes the empty string. To simplify notation, we use $\mathcal{R}_i(\xi)$ to denote $\mathcal{R}_i(P_{\Sigma_i}(\xi))$ when $\Sigma_i \subseteq \Sigma$.

C. Relating Reward Machines to Stochastic Games

To use RMs to specify an RL task, we first need to define a relationship between the stochastic game $\mathcal{G}$ and the task



(a) Reward machine encoding the three-agent crafting task.



(b) Crafting task environment.

Fig. 1: The multi-agent crafting task. In (b) two agents must meet up at the tree, cut the tree down, then transport the log when it appears to the crafting table. The solid black squares are walls. (a) Illustrates the RM encoding this task.

RM $\mathcal{R}$. We define this relationship with the labeling function $L : \mathcal{S} \times U \to 2^{\Sigma}$, which we assume to be known a priori. At each time step, this function takes the joint state of the agents and the reward machine states, and it outputs a collection of reward machine events occurring at that time step. We assume that all events fit into one of two categories: an event attributed to a single agent or a communication event signaling a change in the environment.

For example, in the reward machine in Fig. 1a, events $a1, l1$ correspond to agent 1 arriving and leaving the tree, respectively, and $tr1$ corresponds to agent 1 transporting the log, while events $timber$ and ar signal changes in the environment such as the tree falling down.

D. Reinforcement Learning with Reward Machines

In single agent RL, agents can learn policies to maximize the reward output by a reward machine using q-learning [8]. Agents estimate the value of an action given a state s of an MDP $\mathcal{M}$ and a reward machine state u, effectively learning a separate q-function for each stage of the task encoded in the RM $\mathcal{R}$.

In multiagent RL, agents can learn independently via decentralized q-learning [1]. Similar to the single agent case, agents learn local policies for individual states of the projected reward machine $\mathcal{R}_i$. One key feature of this method is that the learning is completely decentralized; agents are trained to maximize the reward output by their projected reward machines in the absence of their teammates. To accomplish this distributed training, the agents must be aware of any *shared local events* that they have with their

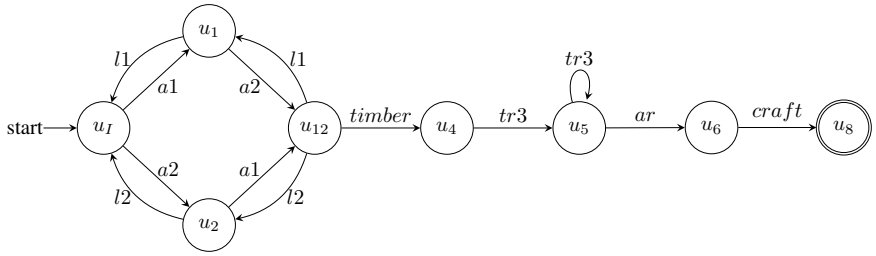


Fig. 2: Strategy RM for the crafting example with a strategy set $\Sigma_S = \{a1, l1, a2, l2, timber, tr3, ar, craft\}$.

teammates. The occurrence of shared events are simulated randomly during training. We refer to [1] for further details.

In general, finding a suitable collection of decentralized reward machines $\mathcal{R}_i$ for a particular team task is non trivial. The focus of this paper is on automatically synthesizing individual reward machines $\mathcal{R}_i = \text{Proj}(\mathcal{R})|_{\Sigma_i}$ that can be used for decentralized learning.

IV. Automatic Decompositions of Team Tasks

In order to decompose the team task we first compute the team's strategy, which describes the specific events that the team plans to execute, in the event that there are multiple ways to complete the task. By specifying such a team strategy we obtain an intermediate reward machine that encodes only the transitions that the team plans on taking. We obtain the team strategy by assigning individual local event sets that represent the roles assigned to individual agents. We formulate the problem as a tree search. At each node of the tree, we check four conditions relating to the feasibility of the associated decomposition. Finally, to select a preferred feasible decomposition, we define a number of criteria used to score each node.

A. Representation of a Team Strategy

Before we can formalize the search for a task decomposition, we develop a representation of the team's *strategy*. We define the team's strategy set $\Sigma_S \subset \Sigma$ as the set of all events the team plans on executing in order to earn a reward. We note that Σ_S will be selected automatically during the task assignment algorithm, described further in Section IV-D. To represent the team's strategy, we define a strategy reward machine, $\mathcal{R}_S$, which encodes only the stages of the task that the team plans to execute, as well as the events that they plan to use to transition between these stages. Formally, $\mathcal{R}_S = \langle U_S, \Sigma_S, u_I, F, \delta_S, \sigma_S \rangle$, where

$$U_S = \{u \in U : F \in reach(u, \mathcal{R})|_{\Sigma_S} \text{ and } u \in reach(u_I, \mathcal{R})|_{\Sigma_S}\}.$$

We define $\delta_S : U_S \times \Sigma_S \to U_S$ as

$$\delta_S(u, e) = \begin{cases} \delta(u, e) & \text{if } u, \delta(u, e) \in U_S \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Similarly, we define $\sigma_S : U_S \times U_S \to \mathbb{R}$ as

$$\sigma_S(u_1, u_2) = \begin{cases} \sigma(u_1, u_2) & \text{if } u_1, u_2 \in U \\ \text{undefined} & \text{otherwise.} \end{cases}.$$

For example, in the team task represented in Fig. 1, an example strategy set is $\Sigma_S = \{a1, a2, timber, tr3, ar, craft\}$, where agent 1 and agent 2 cut down the tree, the tree falls, agent 3 transport the log, arrives at the table, and then crafts the log. The corresponding strategy reward machine $\mathcal{R}_S$ is represented in Fig. 2. Note that this is a much simpler RM that captures only a subset of the original task RM, encoding a single sequence of events that the team plans to execute.

In practice, we obtain the strategy reward machine $\mathcal{R}_S$ from a given strategy set Σ_S by first removing all transitions from $\mathcal{R}$ with an event outside of Σ_S. We then remove any transitions with events in our strategy set that lead to "dead states"—states from which there is no sequence of the remaining events that lead to the final state. Finally, we remove any states that are no longer reachable from our initial state.

We define a strategy set Σ_S along with its corresponding strategy reward machine $\mathcal{R}_S$ to be a *valid strategy* if there exists $\xi \in Tr(\mathcal{R}_S)$ such that $\mathcal{R}(\xi) = 1$. In other words, any valid strategy set Σ_S must allow for a valid sequence of transition on $\mathcal{R}_S$ that results in a team reward. The team strategy depicted in Fig. 2 is valid as the sequence of events $a1\ a2\ timber\ tr3\ ar\ craft$ completes the task.

B. Feasible Decompositions of Strategy Reward Machines

In this section we present conditions to ensure that a task decomposition $\mathcal{R}_1, \ldots \mathcal{R}_n$ of reward machine $\mathcal{R}$ accurately describes the team task and can be successfully applied to decentralized reinforcement learning. Throughout this section, we consider the collection of local event sets $\Sigma_1, \ldots, \Sigma_n \subset \Sigma$ assigned to agents $\alpha_1, \ldots, \alpha_n$, the strategy set $\Sigma_S = \cup_i^n \Sigma_i$, the corresponding strategy reward machine $\mathcal{R}_S$, and resulting task decomposition $\mathcal{R}_i = \text{Proj}(\mathcal{R}_S)|_{\Sigma_i}$.

We now present the first condition:

Condition 1. *Strategy set $\Sigma_S = \Sigma_1 \cup \ldots \cup \Sigma_n$ must be valid.*

The next condition concerns constraints on the task assignment, which we represent as forbidden and required event assignments, and which we assume are provided by the task designer. We define the set of forbidden event assignments $\mathcal{F} \subset \Sigma \times \Lambda$ as follows: if $(e, \alpha_i) \in \mathcal{F}$, then $e \notin \Sigma_i$. For example, in the crafting task agent 1 cannot be assigned event $a2$ as it corresponds to an action agent 2 preforms via the construction of the labeling function. Therefore $(a2, \alpha_1) \in \mathcal{F}$.

Similarly, we define the set of required event assignments: $E \subset \Sigma \times \Lambda$ such that if $(e, \alpha_i) \in E$, then e must be included in the individual event space Σ_i.

Condition 2. *Given forbidden event assignments $\mathcal{F}$, required event assignments E, if $(e, \alpha_i) \in \mathcal{F}$, then $e \notin \Sigma_i$. Similarly, if $(e, \alpha_i) \in E$, then $e \in \Sigma_i$.*

The next condition we present ensures the equivalence between joint execution of the task decomposition $\mathcal{R}_1, \ldots, \mathcal{R}_n$ and the group execution of the strategy reward machine $\mathcal{R}_S$.

Condition 3. *Given the valid strategy set $\Sigma_S = \bigcup_i^n \Sigma_i$ and strategy reward machine $\mathcal{R}_S$, for any sequence of transitions $\xi \in Tr(\mathcal{R}_S)$, $\mathcal{R}_S(\xi) = 1$ if and only if $\mathcal{R}_i(\xi) = 1$, and $\mathcal{R}_S(\xi) = 0$ if and only if $\mathcal{R}_i(\xi) = 0$ for all i.*

This condition says that the individual agents all completing their individual task descriptions $\mathcal{R}_i$ is equivalent to the team completing the team strategy $\mathcal{R}_S$.

For a task decomposition $\mathcal{R}_1, ..., \mathcal{R}_n$, Condition 3 can be verified by checking for bisimilarity between $\mathcal{R}_S$ and the parallel composition of the individual reward machines [1]:

$$\Big\|_{i=1}^{n} \mathcal{R}_i \cong \mathcal{R}_S.$$

This condition can be checked automatically, using the *Hopcroft-Karp* algorithm presented by [27].

While Condition 3 guarantees the correct execution of the team's strategy when all agents correctly complete their planned portion of the task, these guarantees are lost if an agent causes the team to deviate from the strategy RM. In other words, *unplanned* events can force the team into a portion of the original team RM $\mathcal{R}$ that is not included in the strategy RM $\mathcal{R}_S$.

To avoid such unplanned events, we define an augmented reward machine which we use to generate the reward function that the agents will use during training. Formally, given team reward machine $\mathcal{R}$, individual reward machine $\mathcal{R}_i = \text{Proj}(\mathcal{R}_S)|_{\Sigma_i} = \langle U_i, u_I^i, \Sigma_i, \delta_i, \sigma_i, F_i \rangle$ and team strategy Σ_S, an *accident avoidance* RM is defined as the tuple $\mathcal{R}_a = \langle U_i \cup \{u_x\}, \Sigma, u_I^i, F_i, \delta_a, \sigma_a \rangle$. Here, $\delta_a : U_i \times \Sigma \to U_i \cup \{u_x\}$ with

$$\delta_a(u^i, e) = \begin{cases} \delta(u^i, e) & \text{if } \delta_i(u^i, e) \text{ defined,} \\ u_x & \text{if } \delta_i(u^i, e) \text{ undefined} \\ & \quad \text{and } \delta(u, e) \text{ defined} \\ & \quad \text{for } u \in U,\ [u]_\sim = u^i, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

and $\sigma_a : \mathcal{R}_S \times \mathcal{R}_S \cup \{u_x\} \to \mathbb{R}$

$$\sigma_a(u_1, u_2) = \begin{cases} 1 & \text{if } \sigma_S(u_1, u_2) = 1 \\ -1 & \text{if } u_2 = u_x \\ 0 & \text{otherwise.} \end{cases}$$

An accident avoidance reward machine outputs a negative reward whenever an unplanned transition occurs. Fig. 3 illustrates an accident avoidance RM for the strategy RM shown in Fig. 2. The dashed transitions in Fig. 3 lead to u_x, an absorbing state and produces a negative reward. During training, agents will learn to avoid unplanned events: agent 3 will learn to avoid cutting down the tree and agent 2 will learn to avoid transporting the log.

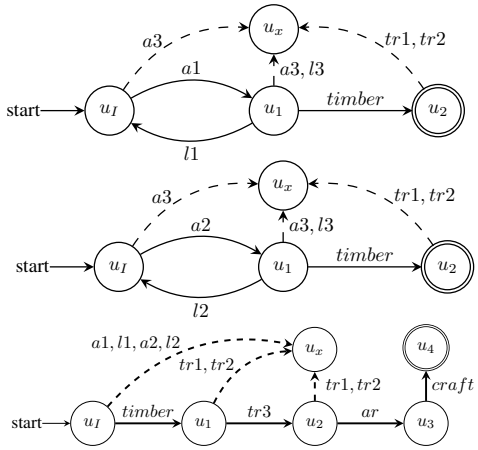


Fig. 3: Accident Avoidance RM for the strategy RM shown in Fig. 2. Dashed transitions result in a negative reward during training as they do not belong to the strategy.

Condition 4. *For a strategy reward machine $\mathcal{R}_S$ and individual reward machines $\mathcal{R}_1, ..., \mathcal{R}_n$. The corresponding accident avoidance reward machines $(\mathcal{R}_i)_a$ must be deterministic.*

We consider a task decomposition $\mathcal{R}_1, \ldots, \mathcal{R}_n$ to be *feasible* if it meets all four conditions outlined in this section.

C. Comparing Task Decompositions

We now introduce various criteria to allow us to quantitatively compare different feasible task decompositions during the tree search, which we describe in Section IV-D. To aid in this discussion, we define the set of task assignments for a given task decomposition $\mathcal{R}_1, ..., \mathcal{R}_n$: $\mathcal{G} = \{(e, \alpha_i) | \alpha_i \in \Lambda, e \in \Sigma_i\}$. This is the set of all event-agent pairs (e, α) such that e is assigned to agent α.

1) Minimizing Necessary Agent Coordination: We consider the number of events assigned to agents. By doing so, we reduce the size of the resulting projected reward machines, and thus reduce the size of the policy that must be learned via RL. Minimizing the number of events assigned to agents' local event sets also limits the number of *shared events* needed to complete a task. Shared events represent moments of coordination between agents: multiple agents must synchronize their understanding of the current stage of the task. By minimizing the number of events assigned to each agent, we remove excess coordination, communication, and training for the agents. To quantify this concept, we give each decomposition a score:

$$\mathcal{E}(\mathcal{G}) = \frac{|(\Sigma \times \Lambda) \setminus (\mathcal{F} \cup \mathcal{G})|}{|(\Sigma \times \Lambda) \setminus (\mathcal{F} \cup E)|}, \tag{1}$$

where $\mathcal{F}$ is the set of all forbidden task assignments and E is the set of all required task assignments. This score

has a lower bound of $\mathcal{E} = 0$, which occurs when the task assignment is as large as possible: $\mathcal{G} = (\Sigma \times \Lambda) \setminus \mathcal{F}$. It has an upper bound of $\mathcal{E} = 1$, which occurs when the task assignment is as small as possible: $\mathcal{G} = E$.

2) Encouraging Fair Task Decompositions: We consider how fairly a decomposition distributes the work between agents. This quality is designed to encourage decompositions that result in a true team effort, as opposed to decompositions in which one agent does the overwhelming majority of the work for the team. To quantify the fairness of a decomposition, we assign a fairness score:

$$\mathcal{X}(\mathcal{G}) = 1 - \frac{\sum_i^n (|\Sigma_i| - l_{avg})}{|\mathcal{G}|}, \tag{2}$$

where $l_{avg} = \sum_i^n |\Sigma_i|/n$ is the average length of event spaces between agents. This score has an upper bound of $\mathcal{X} = 1$, which occurs when all agents have the same number of events assigned to them, and a low score when there is a large difference in the sizes of their event sets.

3) Maximizing Agent Utility in Task Decompositions: Finally, we consider the total utility of a task assignment. We calculate the utility score $\mathcal{U}$ given a utility function H_i for each agent i, where $H_i : \Sigma \to \mathbb{R}$ and $H_i(e) \geq 0$ for all $e \in \Sigma$. This function returns the utility that an agent receives when a particular event is assigned to them. The utility score is given by

$$\mathcal{U}(\mathcal{G}) = \frac{\sum_{(e,\alpha_i)\in\mathcal{G}} H_i(e)}{\sum_{(e,\alpha_i)\in(\Sigma\times\Lambda)\setminus\mathcal{F}} H_i(e)}. \tag{3}$$

This score has an upper bound of $\mathcal{U} = 1$ when the total utility of the agents is equal to the total possible utility of the agents, and has a lower bound of $\mathcal{U} = 0$ when the agents get no utility from the task assignment.

As different teams prioritize these scores differently, the total score is a sum with variable weights. Each task decomposition can then be assigned a total score:

$$s(\mathcal{G}) = w_{\mathcal{E}}\mathcal{E}(\mathcal{G}) + w_{\mathcal{X}}\mathcal{X}(\mathcal{G}) + w_{\mathcal{U}}\mathcal{U}(\mathcal{G}) \tag{4}$$

where $w_{\mathcal{E}}, w_{\mathcal{X}}$ and $w_{\mathcal{U}}$ are the corresponding weights. Using these scores, we qualitatively compare decompositions. Decompositions with higher scores s are a preferred decomposition for the team.

D. Automatic Task Assignment and Decomposition

In this section, we outline the Automatic Task Assignment and Decomposition algorithm (ATAD). To automatically decompose a reward machine, we formulate a tree of task assignments $\mathcal{G}$ and search for a feasible decomposition $\mathcal{R}_1, ..., \mathcal{R}_n$ with corresponding task assignment $\mathcal{G}^*$ that maximizes the total decomposition score $s(\mathcal{G}^*)$.

In the search tree, each node corresponds to a specific task assignment $\mathcal{G}$, and the root corresponds to the largest possible assignment $\mathcal{G}_o = (\Sigma \times \Lambda) \setminus \mathcal{F}$. Every node has exactly two children. One child is the exact replica of the parent node, the other is the parent node with a specific event-agent pair removed. In this way, each level of the tree corresponds to the removal (or inclusion) of a particular event-agent assignment $(e, \alpha_i) \in (\Sigma \times \Lambda) \setminus (E \cup \mathcal{F})$. The order in which each assignment $(e, \alpha) \in (\Sigma\times\Lambda)\setminus(E\cup\mathcal{F})$ is removed is random. The result is a tree that has a leaf for each element in $2^{\mathcal{G}}$.

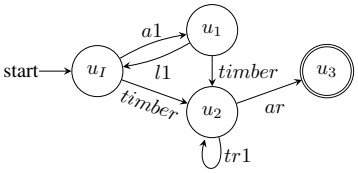


Fig. 4: Subtask RM generated by the baseline decomposition for agent 1.

At each node, ATAD checks if the corresponding task assignment $\mathcal{G}$ satisfies Condition 1. If the node does not satisfy Condition 1, the algorithm stops searching along that branch. If the algorithm reaches a leaf node $\mathcal{G}$, it then compares that node's score $s(\mathcal{G})$ to the best score from any of the previously traversed leaf nodes $s(\mathcal{G}^*)$. If the score is improved, ATAD checks Conditions 3 and 4 to verify whether the decomposition is feasible. If the leaf is feasible, ATAD updates the best task assignment $\mathcal{G}^*$. Note that by the construction of the tree, any leaf will automatically satisfy Condition 2.

If the team only requires a decomposition that is valid, the search can be terminated as soon as it reaches a leaf that satisfies Conditions 3 and 4. Alternatively, if the team has a minimum allowable score for a decomposition, the search can be terminated as soon as a leaf is found that satisfies all conditions and has a score above the desired threshold.

Using the task decomposition generated via ATAD, agents learn policies on their individual accident avoidance reward machines using decentralized q-learning.

V. Numerical Experiments

In this section we present the results of experiments that apply the proposed ATAD algorithm to the decomposition of three-agent and five-agent cooperative MARL tasks. Code to reproduce all experiments is available at: `https://github.com/smithsophia1688/automated_task_assignment_with_rm`.

A. Three-Agent Crafting Task

We begin by considering the crafting task illustrated in Fig. 1b, which we implement as a 10x10 gridworld. Each agent has five available actions: move up, move down, move left, move right, or don't move. We also include stochasticity in the environment transitions: every time an agent selects an action, it has a 2% chance of accidentally slipping and arriving in an unintended neighboring state.

We apply ATAD to the team reward machine depicted in Fig. 1a, and we use the resulting task decomposition for decentralized q-learning. The accident avoidance reward machines that were automatically generated by ATAD for the crafting team task are shown in Fig. 3.

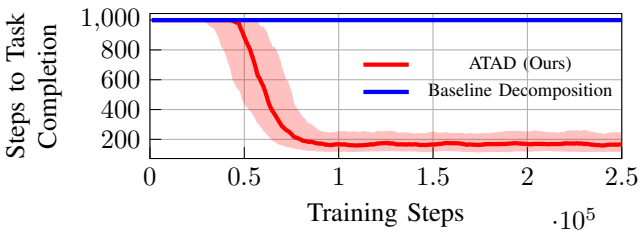


Fig. 5: Numerical results for the three-agent crafting task. We plot the median (over 100 experimental runs) of the number of steps required for task completion during testing. The shaded regions enclose the 25^{th} and 75^{th} percentiles.

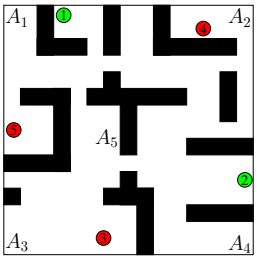


Fig. 6: An illustration of the five-agent buttons task environment. Pressing a red button results in failure of the task.

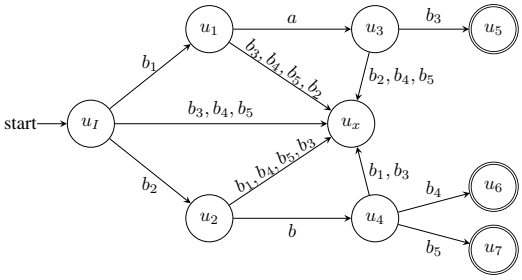


Fig. 7: RM for the five-agent buttons experiment. Transitions a, b signal previously red buttons turning green. The team receives a negative reward in state u_x.

For comparison, we also compute a *baseline* task decomposition and train the agents to complete the resulting subtasks using the same decentralized q-learning procedure as for the ATAD decomposition. This default decomposition is generated by only enforcing the required event assignments from Condition 2. In other words, each agent is assigned all events that are not strictly forbidden. Fig. 4 illustrates agent 1's local reward machine generated by the baseline decomposition. We note this default decomposition meets conditions 4, 2, and 1, but it fails to meet condition 3.

We train the agents using q-learning with a discount parameter $\gamma = .9$ and learning rate of $\alpha = .8$. Agents selected actions using a soft-max exploration policy with temperature parameter $T = .02$ [28].

Each training episode is 1000 time steps long. Periodically throughout training, we test the learned policies and we measure their performance with respect to the team task. Agents do not learn during these team testing executions, and we record the number of time steps that the team requires to complete its task.

Fig. 5 plots the team's performance with respect to its task as a function of the number of elapsed training steps. Specifically, we plot the median (over 100 separate experimental runs) of the number of steps required for team task completion during testing. The shaded regions enclose the 25^{th} and 75^{th} percentiles.

ATAD yields task decompositions that enable efficient decentralized learning. Our approach quickly learns an effective collection of decentralized policies. The team is able to complete the entire task in less than 200 timesteps. The conditions that we enforce during the automated task decomposition procedure guarantee that so long as the agents independently learn policies completing their subtasks, then the team will jointly complete the original task.

Meanwhile, the baseline task decomposition fails to learn policies that complete the task at all. A lack of coordination between the agents, because of the baseline decomposition violating Condition 3, results in the agents becoming "bystanders". Each agent learns to expect that its teammates will complete the relevant subtasks instead of completing them itself. Without proper guarantees that the independently learned policies will jointly result in the team's success, the agents are not properly coordinated and do not learn to complete their required portion as they expect other agents to complete it for them. This demonstrates the importance of the conditions that we formulate for automated task decomposition in MARL.

B. Five-Agent Buttons Task

We now consider a five-agent task in which the agents are required to push a series of buttons in a specific order. Each agent is assigned a specific button that it is able to push—agent i can only push button bi. The team receives a reward of -1 in the event that an agent pushes a button out of order and the training episode is terminated. After two buttons are pushed in order, the team receives a reward of 1 and the episode terminates. Fig. 6 illustrates the task environment.

Using ATAD, the team of agents decomposes the team RM shown in Fig. 7 and automatically selects a given strategy. The experimental results are presented in Fig. 8. The top plot shows the average number of training steps to task completion during the periodic testing of the team's decentralized learning. The bottom plot shows the average discounted reward received during testing. Experiments were carried out with the same parameters as in the previous crafting experiment except with a discount factor $\gamma = .99$ and 15×15 gridworld.

For both the baseline decomposition and ATAD, the agents receive negative reward during the early stages of training because they press buttons out of turn. Both methods eventually learn to stop pressing buttons out of order. However, only the

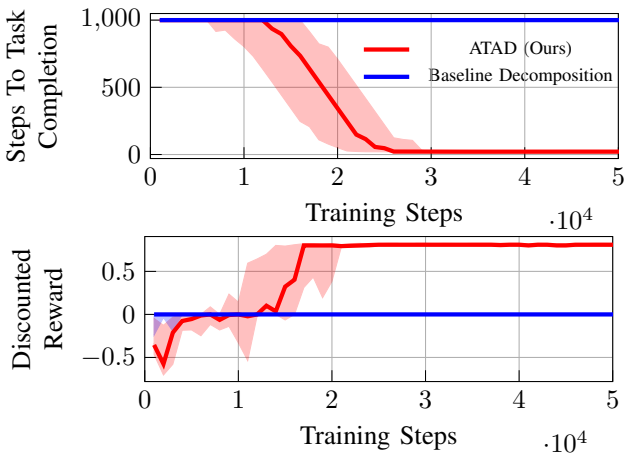


Fig. 8: Numerical results for the five-agent buttons task.

decomposition generated by ATAD results in a collection of policies that eventually complete the task.

VI. Conclusions

We present a framework and algorithms for the automatic decomposition of cooperative multiagent reinforcement learning (MARL) tasks. We develop: 1) conditions that ensure that candidate task decompositions result in behaviors that are consistent with the team task, 2) score functions to compare the candidate decompositions, and 3) an algorithm to automatically search for a decomposition that maximizes a combination of the score functions, while also satisfy the conditions. We experimentally demonstrate that the proposed approach yields task decompositions that can be efficiently learned by decentralized MARL algorithms. Future work will apply the proposed framework to MARL problems involving partial observability and large state-action spaces, and will study how the task decomposition might be iteratively refined during training.

References

[1] C. Neary, Z. Xu, B. Wu, and U. Topcu, "Reward machines for cooperative multi-agent reinforcement learning," in *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS, p. 934–942, International Foundation for Autonomous Agents and Multiagent Systems, 2021.

[2] M. Karimadini and H. Lin, "Guaranteed global performance through local coordinations," *Automatica*, vol. 47, no. 5, pp. 890–898, 2011.

[3] M. Karimadini, H. Lin, and A. Karimoddini, "Cooperative tasking for deterministic specification automata," *Asian Journal of Control*, vol. 18, no. 6, pp. 2078–2087, 2016.

[4] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Decomposition of finite LTL specifications for efficient multi-agent planning," in *Distributed Autonomous Robotic Systems*, vol. 6 of *Springer Proceedings in Advanced Robotics*, pp. 253–267, 2016.

[5] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems," *The International Journal of Robotics Research*, vol. 37, no. 7, pp. 818–838, 2018.

[6] M. O. Karabag, C. Neary, and U. Topcu, "Planning not to talk: Multiagent systems that are robust to communication loss," in *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, AAMAS, p. 705–713, International Foundation for Autonomous Agents and Multiagent Systems, 2022.

[7] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Reward machines: Exploiting reward function structure in reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 73, pp. 173–208, 2022.

[8] R. T. Icarte, T. Klassen, R. Valenzano, and S. McIlraith, "Using reward machines for high-level task specification and decomposition in reinforcement learning," in *International Conference on Machine Learning*, pp. 2112–2121, 2018.

[9] R. Toro Icarte, E. Waldie, T. Klassen, R. Valenzano, M. Castro, and S. McIlraith, "Learning reward machines for partially observable reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[10] Z. Xu, I. Gavran, Y. Ahmad, R. Majumdar, D. Neider, U. Topcu, and B. Wu, "Joint inference of reward machines and policies for reinforcement learning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, pp. 590–598, 2020.

[11] 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*, vol. 70, pp. 1031–1116, 2021.

[12] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," 2019.

[13] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. de Cote, "A survey of learning in multiagent environments: Dealing with non-stationarity," 2017.

[14] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 33, no. 6, pp. 750–797, 2019.

[15] L. Matignon, G. J. Laurent, and N. L. Fort-Piat, "Independent reinforcement learners in cooperative markov games: A survey regarding coordination problems," *The Knowledge Engineering Review*, vol. 27, no. 1, pp. 1–31, 2012.

[16] M. Tan, "Multi-agent reinforcement learning: Independent versus cooperative agents," in *Proceedings of the 10th International Conference on Machine Learning*, pp. 330–337, 1993.

[17] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *International Conference on Machine Learning*, pp. 4295–4304, PMLR, 2018.

[18] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi, "QTRAN: learning to factorize with transformation for cooperative multi-agent reinforcement learning," in *International Conference on Machine Learning*, pp. 5887–5896, PMLR, 2019.

[19] A. Mahajan, T. Rashid, M. Samvelyan, and S. Whiteson, "Maven: Multi-agent variational exploration," in *Advances in Neural Information Processing Systems*, pp. 7613–7624, 2019.

[20] C. Guestrin, M. Lagoudakis, and R. Parr, "Coordinated reinforcement learning," in *International Conference on Machine Learning*, vol. 2, pp. 227–234, 2002.

[21] E. Van der Pol and F. A. Oliehoek, "Coordinated deep reinforcement learners for traffic light control," *Proceedings of Learning, Inference and Control of Multi-Agent Systems (at NIPS 2016)*, 2016.

[22] C. Boutilier, "Planning, learning and coordination in multiagent decision processes," in *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pp. 195–210, Morgan Kaufmann Publishers Inc., 1996.

[23] B. G. León and F. Belardinelli, "Extended markov games to learn multiple tasks in multi-agent reinforcement learning," 2020.

[24] I. ElSayed-Aly and L. Feng, "Logic-based reward shaping for multi-agent reinforcement learning," *CoRR*, vol. abs/2206.08881, 2022.

[25] A. Velasquez, B. Bissey, L. Barak, D. Melcer, A. Beckus, I. Alkhouri, and G. Atia, "Multi-agent tree search with dynamic reward shaping," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, pp. 652–661, 2022.

[26] J. Eappen and S. Jagannathan, "DistSPECTRL: Distributing specifications in multi-agent reinforcement learning systems," *arXiv preprint arXiv:2206.13754*, 2022.

[27] F. Bonchi and D. Pous, "Checking NFA equivalence with bisimulations up to congruence," *ACM SIGPLAN Notices*, vol. 48, no. 1, pp. 457–468, 2013.

[28] A. D. Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for q-learning in random stochastic mazes," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, IEEE, 2016.