

Learning Task Automata for Reinforcement Learning in Multi-Agent Systems

Joseph Lawson

University of Oxford

Trinity 2023



Contents

1	Introduction	4
1.1	Related Research	5
1.2	Contributions	6
2	Setup	7
2.1	Labelled Markov Decision Process	8
2.2	Task Automaton	10
2.3	Labelled Markov Game	10
2.4	Product MDP	12
2.5	Partially-Observable Markov Decision Process	13
3	Overview of Single Agent Learning Pipeline	13
3.1	Step 1: Learn Product MDP	14
3.2	Step 2: Distill TA from a learnt product MDP	15
3.3	Step 3: Minimise the TA and remove Environmental Bias	16
4	How to extend to Multi-Agent Settings	18
4.1	Outline of Multi-Agent Adaptation	18
4.2	Local Trajectories, Traces and TAs	20
4.3	Local Alphabets	21
4.4	Conditions on Local TAs Necessary for Multi-Agent Q-Learning Algorithm	21
4.5	The Learnt TAs	25
4.6	Attainable TA	27
4.7	Decomposable Labelling function	29
4.8	Agent Communication	31
4.9	Multi-Agent TA Learning Algorithm	31
4.10	Local Reward Functions	32
5	Experimental Results	33
5.1	Setup	33

5.2	Performance on Tasks and Learnt TAs	34
5.3	Scalability	36
5.4	Events with Low Probability	38
5.5	Local Reward Functions	39
6	Conclusion	40
6.1	Reflections	41
6.2	Future Work	41

1 Introduction

The effectiveness of Reinforcement learning for Markovian tasks has been widely demonstrated over the last decade [18]. Despite this, standard approaches to Reinforcement Learning perform poorly at learning an optimal policy for tasks with sparse, non-Markovian reward signals [14]. A policy is a probability distribution over the possible set of actions for a given agent, with an optimal policy maximising the reward an agent receives. A Markovian task is where the reward function only depends on the most recent action, for example, if an agent has just moved from the stairs to the door in one step. There are many real-world tasks that are both sparse and non-Markovian, specifically those that require some sequence of sub-tasks to be completed before a reward is given. For example, say the task is to exit the room but the key must be found first before going to the door. In this case, the reward is non-Markovian as the agent must perform an action to get the key before then performing an action to go to the door, at which point it is rewarded.

Recently there has been a focus on creating new techniques to adapt existing Reinforcement Learning methods to tasks with sparse, non-Markovian rewards. I will focus on methods which use Linear Dynamic Logic on finite traces to express the task the agent needs to complete to gain a reward. Within this, there are two main areas of focus. First, methods that focus on using the Linear Dynamic Logic representation of the task in order to learn an optimal policy. Second, taking an agent with no prior knowledge of its task and learning the linear dynamic logic representation of the task by exploring its environment. Both these techniques have had recent success and have been applied to a range of single-agent settings, however, only the learning of an optimal policy given a task has been applied to multi-agent settings.

In this report, I describe in Section 2 the type of agents we are going to consider, and how they and their tasks are represented in both the single and multi-agent cases. Then, in Section 3, I present an existing pipeline that learns tasks in the single-agent case. This existing pipeline is used as the basis for my multi-agent approach.

In Section 4, I propose, for the first time, how Linear Dynamic Logic representations of tasks can be learnt in a multi-agent setting and show how they can be used to synthesise optimal policies. I provide a list of assumptions on the multi-agent system that are sufficient for the representations to be learnt, as well as providing extensions to my technique that can allow my method to be applied to a wider range of tasks. Finally, in Section 5, I demonstrate experimentally the success of my approach and compare its performance on a range of different tasks.

1.1 Related Research

There are various approaches to efficiently learning optimal policies in settings with sparse, non-Markovian reward signals. One such approach utilises Linear Dynamic Logic (LDL) to assist in learning an optimal policy [3, 6, 9, 11–13, 15, 19]. The agent explores a Markov Decision Process (MDP) guided by fulfilling an LDL property, often represented using a Task Automaton (also known as a Reward Machine), which is a type of Deterministic Finite Automata (DFA). This representation allows a non-Markovian task to be broken down into a series of Markovian sub-tasks that are represented as transitions in the Task Automaton (TA).

Recent work has extended this approach to multi-agent settings [16, 21], which greatly increases the scope of problems that can be tackled using TAs. If we restrict our attention to non-competitive multi-agent settings, Neary et al.[16] demonstrated that it is more efficient to define a local TA for each agent, where the composition of these local TAs is equivalent to the TA for the whole system, than to use the overall TA. A local TA encodes only the part of the task that a specific agent contributes towards, allowing the methods for learning optimal policies for single agents with non-Markovian rewards to be efficiently applied to multi-agent settings.

In single-agent settings, a set of similar extensions have been developed where the agent initially has no knowledge of the tasks and learns the TA through environment exploration [1, 7, 20, 22]. This makes a wider range of problems tractable, specifically where the full

task specification is not known by the human in advance.

Towards the end of writing this report, a paper which attempts to learn TAs for multi-agent systems appeared online [4]. It uses inductive logic programming to learn sets of local TAs from local observation sets, and this is demonstrated for two hand-picked examples. They also combine their learnt local TAs with the method outlined in [16] to learn an optimal policy. However, there is no attempt to justify or provide conditions for the success of the algorithm on any example outside the two they considered. Secondly, as they only focus on two simple examples it is unclear how their approach would work when faced with the problems discussed in this report. Therefore it is not possible to compare the theoretical advantages of one approach against the other. Also, the inductive logic programming algorithm they use is demonstrated to have severe scalability issues when learning complex TAs. Finally, their code is not yet available, and no run times are given within the paper, thus it is impossible to experimentally compare my approach against theirs. As a result of the inability to compare theoretically or practically, I will not discuss this approach any further.

1.2 Contributions

The contributions of this master’s project are as follows:

- In Section 4, I adapt a pipeline outlined in [1] for learning TAs for single-agent systems so that it can be used to learn local TAs for each agent in a multi-agent system in parallel. The interleaving of these local TAs is equivalent to the global TA for the entire system.
- I show that these learnt TAs can be used as part of the method first described in [16] to synthesise an optimal policy for the multi-agent system.
- Throughout Section 4, I discuss how both the structure of the global task and the communication between the agents impact if we can learn local TAs. I also provide alternative formulations that could be used to solve some problems that my pipeline cannot solve.

- In Section 5, I demonstrate practically how the contributions in this work allow a far wider range of multi-agent tasks to be solved in this way compared to existing methods.

My contributions above solve previously unsolved problems in the literature, specifically learning local TAs for multi-agent systems, and using learnt local TAs to learn an optimal policy in a multi-agent system.

2 Setup

In a single-agent system, the agent’s interaction with an environment can be viewed at multiple levels of abstraction. Usually, the agent’s dynamics within an environment are modelled using a *Markov Decision Process* (MDP). In order to recognise higher-level features of an environment a *labelled* MDP (see Definition 1) can be used. This uses a set of atomic propositional variables from some alphabet of labels AP which have a truth value at every state and is encoded using some labelling function L . Task specifications are sequential compositions of the higher level features and can be encoded as a TA (see Definition 2) which is a DFA that graphically represents the task that an agent needs to complete in order to gain a reward. It is possible to combine the MDP and the TA into a Product MDP (see Definition 4), if we assume the MDP and TA to be unknown then, as shown in [1], this structure is analogous to a partially observable MDP (POMDP) (see Definition 5). This then allows us to learn the transitions of the MDP, as well as the unknown TA.

A multi-agent system can be modelled using a Markov Game, or when recognising higher-level features of the environment a labelled Markov Game (See Definition 3), which is an extension of the idea of a labelled MDP. A labelled Markov game can also have an associated task specification that is encoded as a TA.

2.1 Labelled Markov Decision Process

When considering a single agent in an environment with some non-Markovian reward function we need to provide a way to model this agent. Specifically, we need to model the states it can be in, the actions it can take in each state, along with the probability a given action takes you from one state to another. Also, we need to specify a function that returns the reward an agent will receive given its history of states and actions. It is also necessary to have a set of possible higher-level features (also known as events) and a mapping that labels each of the agent’s states with a set of these events. We can formalise this into a labelled Markov Decision Process according to the following definition.

Definition 1. A **labelled Markov Decision Process (MDP)** is a tuple

$\mathcal{M} = (\mathcal{S}, \mathcal{A}, s_0, P, R, \gamma, \mathcal{AP}, L)$, where \mathcal{S} is a finite set of states, \mathcal{A} is a finite set of actions, $s_0 \in \mathcal{S}$ is the initial state, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the probability distribution for transitions, $R : (\mathcal{S} \times \mathcal{A})^+ \times \mathcal{S} \rightarrow \mathbb{R}$ is a (non-Markovian) reward function, $\gamma \in [0, 1)$ is the discount factor, \mathcal{AP} is a finite alphabet of atomic propositions (labels), and $L : \mathcal{S} \rightarrow 2^{\mathcal{AP}}$ is a labelling function.

A trajectory is the sequence of states and actions in our system $(s^0, a^0, s^1, \dots, a^{n-1}, s^n)$, this is accompanied by a reward sequence (r^0, r^1, \dots, r^n) and a trace of labels $(\ell^0, \ell^1, \dots, \ell^n)$, where $\forall t \leq n. L(s_t) = \ell_t$. The task of the agent is to find some policy π that maximises the expected value of the sum of discounted rewards $E_\pi[\sum_{t=0}^{\infty} \gamma^t r^t]$.

In this work, I will focus on sparse reward environments, specifically tasks with a reward sequence of $(0, 0, \dots, 1)$ where a non-zero reward is only achieved when the overall task is completed. This is a non-Markovian reward function because it relies on the entire state-action trajectory up to time t , whereas a Markovian reward function depends only on the last transition.

Example 1. An example of the above is the grid world in Figure 1a . This has an RL agent in an MDP environment, the task is to collect coffee ☕ , take it to the couch 🛋 , turn on the TV 📺 and then ascend the stairs 🏠 . The carpet 🧶 and book 📖 do not affect the agent achieving its goal. The arrow indicates the initial state[1].

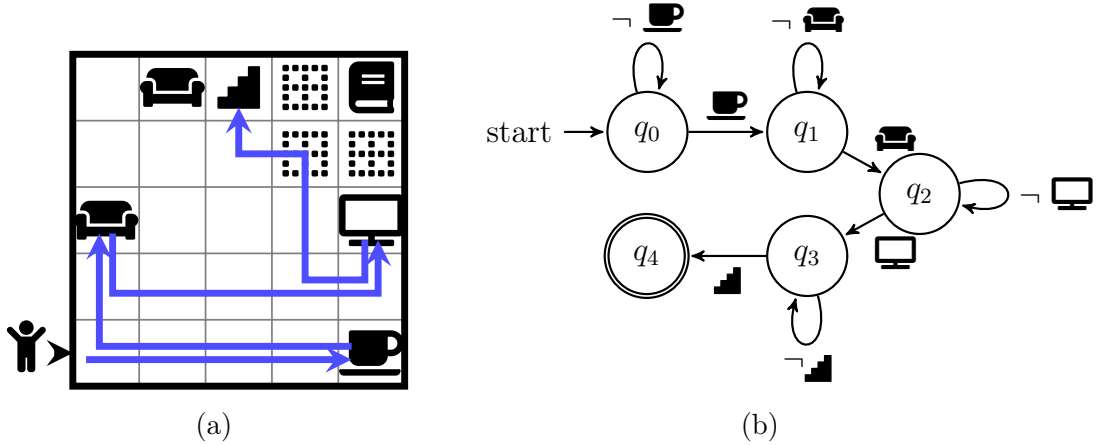


Figure 1: (a) The labelled MDP environment and (b) 5 state TA representing a task specification for Example 1.

The MDP environment for Example 1 is in Figure 1a. Each square represents a state belonging to \mathcal{S} with s_0 being the bottom left square. In this and all future grid world examples, the set of actions \mathcal{A} for an agent is to move into the adjacent up, down, right and left squares. Thus the probability, defined by P , of moving from a square to an adjacent square given the correct action is 1. If an agent attempts to do an impossible action they will just remain where they are, this is equivalent to saying that if an agent attempts to move to a different square via an incorrect action or move to a non-adjacent square, then the probability of this transition is 0. The set of atomic propositional variables $\mathcal{AP} = \{\text{coffee}, \text{car}, \text{stairs}, \text{computer}, \text{stairs}\}$, are the high-level features of the environment, the states in the figure that are not explicitly labelled are labelled with $L(s) = \emptyset$. An example trace is $(\emptyset, \emptyset, \emptyset, \emptyset, \text{coffee})$ which is the trace for the trajectory which moves right to the coffee from the initial state. R is defined such that it is equal to 1 when the state action trajectory of the agent at that point has completed its task.

We consider tasks where the only reward is upon completion. Clearly, this example cannot be expressed with a Markovian reward function over the original state-space, as it is impossible to go between for example labels coffee and car in one-time step. Therefore, when the agent is trying to learn an optimal policy in this example it would have to plan using a memory of its previous states. This is where the labelling function comes in. To avoid the ‘curse of dimensionality’, the agent can instead remember only the traces of the

higher-level features, which allows the agent to keep track of the completion of sub-tasks without having to keep a memory of the entire state-action trajectory.

2.2 Task Automaton

Linear Dynamic Logic, when restricted to finite trajectories (LDL_f) can be used to express a specification for a task or a constraint on a plan. This is used to solve the problem of sample inefficiency in non-Markovian RL environments. LDL_f can be expressed as a DFA [8] thus we will define a Task Automaton as a DFA that encodes the task specification for our MDP.

Definition 2. A **Task Automaton (TA)** is a tuple $\mathcal{A} = (\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{F})$, where \mathcal{Q} is a finite set of states, $q_0 \in \mathcal{Q}$ is the initial state, $\Sigma = 2^{\mathcal{A}^{\mathcal{P}}}$ is a finite alphabet inherited from the labelled MDP, $\delta : \mathcal{Q} \times \Sigma \rightarrow \mathcal{Q}$ is a transition function and $\mathcal{F} \subseteq \mathcal{Q}$ is the set of accepting states.

A TA defines a task specification on labels for an agent. Each move in the MDP matches a transition in the TA according to the label of the new state of the MDP. Therefore, the state of a TA describes the progression of the agent through the task so far and thus serves as the required ‘memory’ of state trajectories mentioned earlier. All labels not on an outgoing edge are implicitly on a self-loop. The accepting states of the TA mark the satisfaction of the specification which is when the agent will receive its reward. The TA for the task specification in Example 1 is represented in Figure 1b.

2.3 Labelled Markov Game

In a multi-agent setting, we need a way to model all the agents in a given environment, we can do this by defining a similar structure to an MDP that takes into account the fact that we now have multiple agents. Unlike in an MDP, we need to describe how the transitions of the agents are impacted by the transitions and states of the other agents. Also, we must consider how the labelling function is defined as there is no longer a simple mapping between individual states and labels, instead, we must define a labelling function on the







combination of all the agents' states. We will capture a set of agents in a multi-agent environment using a labelled Markov Game as described below.

Definition 3. A **labelled Markov Game** is a tuple $\mathcal{G} = (\mathcal{I}, \mathcal{S}, \mathcal{A}, P, R, \gamma, \mathcal{AP}, L)$, where $\mathcal{I} = \{1, \dots, n\}$ is a finite set of agents, $\mathcal{S} = \mathcal{S}_1 \times \dots \times \mathcal{S}_n$ is the finite set of joint states where \mathcal{S}_i is the finite set of states for each agent i . $\mathcal{A} = \mathcal{A}_1 \times \dots \times \mathcal{A}_n$ is a finite set of actions where \mathcal{A}_i is the finite set of actions for agent i . $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition probability distribution. $R : (\mathcal{S} \times \mathcal{A})^+ \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward function. $\gamma \in [0, 1)$ is the discount factor. $L : \mathcal{S} \rightarrow 2^{\mathcal{AP}}$ is a labelling function.

A local policy $\pi_i : \mathcal{S}_i \times \mathcal{A}_i \rightarrow [0, 1]$ is a probability distribution over state-action pairs, that is at a time step t agent i is in state $s_i^t \in \mathcal{S}_i$ and the agent chooses action $a_i^t \in \mathcal{A}_i$ with probability $\pi_i(a_i^t | s_i^t)$. The agent then transitions to a new state $s_i^{t+1} \in \mathcal{S}_i$ with some probability dependent on the states and actions of all other agents, and all agents receive a reward $r^t = R(s^0, a^0, \dots, s^t, a^t, s^{t+1})$. The joint policy of all the agents is $\pi = (\pi_1, \dots, \pi_n)$.

A trajectory is the sequence of states and actions in our system $(s^0, a^0, s^1, \dots, a^{n-1}, s^n)$, this is accompanied by a reward sequence (r^0, r^1, \dots, r^n) and a trace of labels $(\ell^0, \ell^1, \dots, \ell^n)$, where $\forall t \leq n. L(s_t) = \ell_t$. Generally, the task of the agents is to find some joint policy π that maximises the expected value of the sum of discounted rewards $E_\pi[\sum_{t=0}^{\infty} \gamma^t r^t]$. In this work, I will focus on tasks with a reward sequence of $(0, 0, \dots, 1)$ where a non-zero reward is only achieved when the overall task is completed.

Note that we are working within a cooperative setting, which is because all agents receive the same reward. Also, note that they receive a reward at the completion of the global task, so no agent is being rewarded for a subtask.

Example 2. *Figure 2a is an example of the above. This has 2 agents in a labelled Markov Game environment. There are two agents Red and Blue, the task is for Red to collect coffee , and take it to the red couch . Then Blue needs to collect toast  and take it to the blue sofa , then one of the agents needs to turn on the black TV , and then finally they must meet at the green stairs . The arrows indicate the start states, blue labels only appear when Blue is at that state, red labels only appear when Red is at*

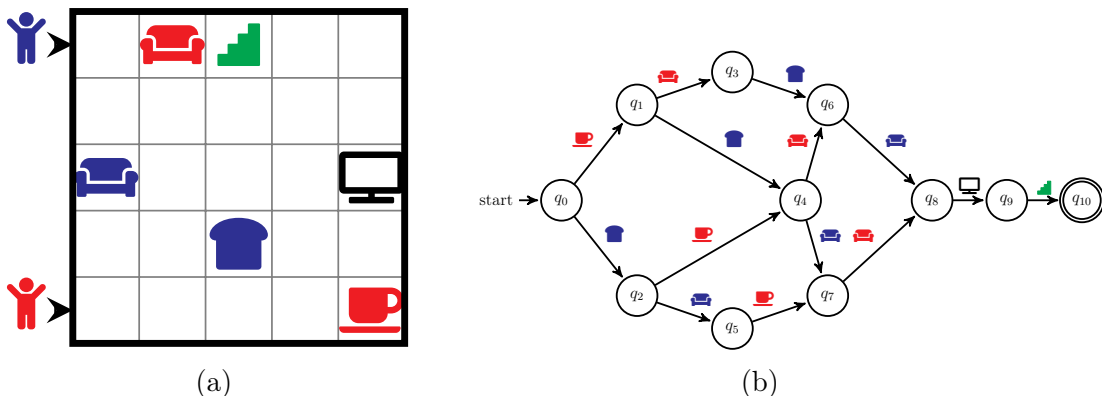


Figure 2: (a) The labelled Markov Game environment and (b) 11 state TA representing a task specification for Example 2.

that state, black labels can be accessed by either agent, green labels require both agents to be at the state.

In example 2, the local states and actions for each agent are the same as the set of states in example 1. In this particular example, the transition probabilities for each agent are independent of each other and can be defined on their local states analogously to in example 2. The overall transition probability function P is the product of the local transition probability functions. R is defined as 1 only when the task has been satisfied and $\mathcal{AP} = \{\text{cup, sofa, shirt, sofa, staircase, monitor}\}$.

2.4 Product MDP

We have defined both MDPs and TAs which are used to define an agent’s environment and its progress through a given task. For the purposes of learning TAs, it is useful to consider the product of an MDP without rewards and a TA. Intuitively you can see this as an MDP but with each state labelled with the current TA state, then the transitions between product states are only possible if there is a transition between MDP states for that action and a transition between TA states for the label of the new MDP state.

Definition 4. Given an MDP without rewards $\mathcal{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ and a TA $\mathcal{A} = (\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{F})$, the **product MDP** is the structure $\mathcal{M} \otimes \mathcal{A} = (\mathcal{S}^\otimes, \mathcal{A}, s_0^\otimes, P^\otimes, R^\otimes)$,

where $\mathcal{S}^\otimes = \mathcal{S} \times \mathcal{Q}$, $s_0^\otimes = \langle s_0, q_0 \rangle$ is the initial state pair; $P^\otimes : \mathcal{S}^\otimes \times \mathcal{A} \times \mathcal{S}^\otimes \rightarrow [0, 1]$ is the transition probability function, where $P^\otimes(\langle s_j, q_j \rangle \mid \langle s_i, q_i \rangle, a_i) = P(s_j \mid s_i, a_i)$ if $q_j = \delta(q_i, L(s_j))$ and 0 otherwise; and $R^\otimes : \mathcal{S}^\otimes \times \mathcal{A} \times \mathcal{S}^\otimes \rightarrow \{0, 1\}$ is a reward function such that $R^\otimes(\langle s_t, q_t \rangle, a_t, \langle s_{t+1}, q_{t+1} \rangle) = \chi_{\mathcal{F}}(q_{t+1})$, where $\chi_{\mathcal{F}}$ is the indicator function for the set of accepting states in the TA. In other words, in the product MDP, a non-zero reward is obtained if and only if the agent transitions to an *accepting state* of the product MDP, i.e., those related to an accepting TA state [1].

2.5 Partially-Observable Markov Decision Process

When an agent with no knowledge of its task is exploring the environment then it has no knowledge of its current product MDP state. All the agent observes is the current MDP state and whether the agent has received a reward or not. Thus the full product state is hidden, and the only observations are the rewards. We can model this in the following way.

Definition 5. A Partially-Observable Markov Decision Process(POMDP) is a tuple $\mathcal{P} = (\mathcal{S}, \mathcal{A}, s_0, P, R, \gamma, \mathcal{O}, Z)$ where $(\mathcal{S}, \mathcal{A}, s_0, P, R, \gamma)$ is an MDP, \mathcal{O} is an observation set, and $Z : \mathcal{O} \times \mathcal{S} \rightarrow [0, 1]$ is the observation probability function, where $Z(o|s)$ is the probability that observation $o \in \mathcal{O}$ is seen when the agent is at the hidden state $s \in \mathcal{S}$ [1].

Restricted to the cases we are considering, the observations are the rewards 0 or 1, and we observe 0 with probability 1 when we are not in an accepting state of the TA. Conversely, we observe 1 with probability 1 when we are in an accepting state of the TA.

3 Overview of Single Agent Learning Pipeline

Before considering multi-agent examples, I will present the existing algorithm for TA learning in the single-agent case. The algorithm learns an MDP and a TA in completely unknown environments. The pipeline is outlined in Algorithm 1 which was first presented in [1].

Algorithm 1 Learning a TA in an unknown labelled MDP

Input: put agent into an (unknown) labelled MDP \mathcal{M}

Output: TA \mathcal{A}^* that represents a task specification

- 1: OBSSEQ \leftarrow collect episodes of corresponding trajectories, traces, and reward sequences.
 - 2: **function** LEARNPRODUCTMDP(OBSSEQ) ▷ Step 1
 - 3: Use a HMM/POMDP learning algorithm with OBSSEQ to learn and return an estimate of the transition probability distribution \hat{P}_π of the spatial MC $\hat{\mathcal{M}}_\pi$.
 - 4: Use a HMM/POMDP learning algorithm initialised with \hat{P} as an inductive bias and trained with OBSSEQ to learn and return an estimate of the transition probability distribution \hat{P}^\otimes of the product MDP $\hat{\mathcal{M}} \otimes \hat{\mathcal{A}}$.
 - 5: **function** DISTILTA($\hat{\mathcal{M}} \otimes \hat{\mathcal{A}}$) ▷ Step 2
 - 6: Determine $\hat{\mathcal{M}} \otimes \hat{\mathcal{A}}$ using Cone Lumping method to return the MDP-restricted TA $\hat{\mathcal{A}}_{\hat{\mathcal{M}}}$
 - 7: **function** POSTPROCESS($\hat{\mathcal{A}}_{\hat{\mathcal{M}}}$) ▷ Step 3
 - 8: Remove environmental bias and minimise $\hat{\mathcal{A}}_{\hat{\mathcal{M}}}$
-

3.1 Step 1: Learn Product MDP

The agent’s interactions with the environment are fully modelled by an underlying product MDP $\mathcal{M} \otimes \mathcal{A}$ (Definition 4); however, as the agent explores the environment the states of the product MDP are not fully observable. That is, the agent observes the MDP states, the labels and rewards but not the TA states. Therefore, in order to learn the product MDP the agent’s interactions with the environment are viewed as a partially-observable Markov Decision Process (Definition 5). Let the product MDP $\mathcal{M} \otimes \mathcal{A} = (\mathcal{S}^\otimes, \mathcal{A}, s_0^\otimes, P^\otimes, R^\otimes)$, then the POMDP $\mathcal{P}^\otimes = (\mathcal{S}^\otimes, \mathcal{A}, s_0^\otimes, P^\otimes, R^\otimes, \mathcal{O}, Z)$. With $\mathcal{O} = S \times \{0, 1\}$ and $Z : \mathcal{S}^\otimes \rightarrow \mathcal{O}$. The intuition is that at each time step the agent observes the current MDP state and also whether the task is completed or not. Thus all the agent observes is the tuple $(s_t, Z(s_t^\otimes))$. Where $s_t^\otimes = (s_t, x_t)$ and $Z(s_t^\otimes) = 1$ if and only if x_t belongs to the accepting set of states in the underlying TA, and $Z(s_t^\otimes) = 0$ otherwise. Choosing some policy π induces a partially observable Markov chain \mathcal{P}_π^\otimes , also known as a Hidden Markov Model.

Many existing algorithms solve the problem of learning HMMs/POMPDs, in this pipeline the Baum-Welch algorithm is used [5]. The Baum-Welch algorithm is used twice, first to learn an estimate \tilde{P} of the MDPs transition distribution. Then this is used as an

inductive bias for learning an estimate of the transition distribution for the product MDP \tilde{P}^\otimes . We can formalise the result of the Baum-Welch algorithm as the following: $(\tilde{P}_\pi^\otimes, \tilde{Z}) = \underset{(P^\otimes, Z)}{\operatorname{argmax}} Pr(O|(P^\otimes, Z))$. This is the transition distribution for some product MDP in the equivalence class $[\mathcal{M} \otimes \mathcal{A}]$.

Two product MDPs $[\mathcal{M}_1 \otimes \mathcal{A}_1]$ and $[\mathcal{M}_2 \otimes \mathcal{A}_2]$ with the same action spaces $\mathcal{A}_1 = \mathcal{A}_2$ belong to the same equivalence class if they are observationally equivalent. They are observationally equivalent if for any observation sequence, any action trajectory, and all $\tau \in \mathbb{N}$, it holds that $Pr^{\mathcal{P}_1^\otimes}[(o_t)_{t=0}^\tau|(a_t)_{t=1}^\tau] = Pr^{\mathcal{P}_2^\otimes}[(o_t)_{t=0}^\tau|(a_t)_{t=1}^\tau]$, where $P_i^\otimes = (\mathcal{M}_i \otimes \mathcal{A}_i, \mathcal{O}, Z)$.

In summary, given state trajectories and reward sequences for an unknown product MDP $\mathcal{M} \otimes \mathcal{A}$ the algorithm learns an estimate $\tilde{\mathcal{M}} \otimes \tilde{\mathcal{A}}$ of some MDP in the equivalence class $[\mathcal{M} \otimes \mathcal{A}]$.

3.2 Step 2: Distill TA from a learnt product MDP

From step 1, the agent has both an estimate \tilde{P} for the transition distribution and also the digraph of $\mathcal{M} \otimes \mathcal{A}$. The first step is to use a fully mixed policy π to compute a product Markov chain $\tilde{\mathcal{M}}_\pi \otimes \tilde{\mathcal{A}}$. If we label each edge in the digraph underlying $\tilde{\mathcal{M}}_\pi \otimes \tilde{\mathcal{A}}$ with the label of the MDP state then it results in an NFA whose accepting states are those where a reward of 1 is obtained. An example of the NFA that is produced is in Figure 3 [1].

After the NFA has been constructed it can be converted into a DFA that will have the same language as the TA \mathcal{A} . ¹Cone Lumping is used for constructing the DFA from the NFA and was first proposed in [1]. Cone Lumping exploits the fact that any outgoing transition from a state $(s, q) \in S^\otimes$ to $(s', q') \in S^\otimes$ in the underlying NFA that has the same label as another outgoing transition from $(s, q) \in S^\otimes$ must be a transition to a

¹In actuality there is a slight restriction here. The DFA we learn can only be the sub-graph of \mathcal{A} that is covered by traces that are attainable in the MDP. Thus the language of the learnt DFA is not necessarily exactly the language of the full TA \mathcal{A} but instead the language of the parts of \mathcal{A} that are accessible via the MDP. However, if a task specification includes inaccessible parts then it does not matter if these parts are not learnt as they would never be used for synthesising an optimal policy.

product state with the same TA state $(s'', q') \in S^\otimes$. Cone lumping can determinise any product MDP in the worst-case exponentially less time than the Rabin and Scott subset construction method [1, 17]. It does so in $O(|S^\otimes|^3)$ compared to $O(2^{|S^\otimes|})$.

3.3 Step 3: Minimise the TA and remove Environmental Bias

Some learnt TA \mathcal{A}' may not necessarily be a minimal TA. Standard methods for minimising DFAs can be used such as Hopcroft’s algorithm [10]. However, it is also possible that the TA \mathcal{A}' is not minimal due to environmental bias. Environmental bias is when a learnt TA \mathcal{A}' contains a label that is not necessary for the completion of the task, i.e. it is not present at that position in the actual TA \mathcal{A} . This can occur when even though the label isn’t necessary to fulfil the task specification it is impossible to complete the task without encountering this label. For example, say an agent needs to get to a state with a label \blacksquare but all the surrounding states are labelled with \blacksquare . The algorithm learning the TA cannot tell the difference between the importance of \blacksquare and \blacksquare , thus the algorithm assumes \blacksquare is part of the task specification as it always appears directly before \blacksquare in any trace that leads to a reward.

This can be somewhat overcome by a post-processing step. In this step, many TAs \mathcal{A}' such that $\mathcal{M} \otimes \mathcal{A}' \in [\mathcal{M} \otimes \mathcal{A}]$ are considered. Then the TA that requires the smallest alphabet is chosen. This is done by considering each possible label l and guessing that it is irrelevant to the true TA \mathcal{A} . Then merge any states in \mathcal{A}' that have an l transition between them. Then check if this resultant TA still returns the correct output on all the traces in the observation sequence. If so we know the label can be removed. This step only works to remove labels that are irrelevant to the entire TA and does not work on labels that are necessary somewhere in the TA and irrelevant in other places.

4 How to extend to Multi-Agent Settings

The simplest method for extending the TA learning algorithm to a multi-agent setting involves modelling it as if there was only one agent. This is achieved by encoding the transition probabilities of each individual agent into a product transition probability, inducing a Markov chain M that represents the states of all the agents, and estimating the product MDP $\mathcal{M} \otimes \mathcal{A}$. However, there are several issues with this approach. Firstly, the size of the learned TA can be relatively large. For instance, in Example 2 the complexity of the TA is high. This is a significant concern because, as demonstrated in [1], the TA learning algorithm’s scalability is affected by the size of the TA being learned, experimentally it scaled exponentially with the size of the TA. Secondly, this approach assumes that all agents are controlled by a single entity and that all agents see all labels. This assumption is not always valid, as the ways agents communicate and share information differ depending on the example. Finally, the ability to parallelize the learning of the product MDP or TA is limited since everything is treated as a single unit. As a result, this greatly increases the algorithm’s run time, particularly for examples with many agents. Therefore, it is clear that an alternative method of learning TAs in a multi-agent setting is necessary. In particular, a new method is required that more carefully considers the differences between single and multi-agent problems.

4.1 Outline of Multi-Agent Adaptation

The proposed approach involves the splitting of the learning of task automata (TA) for a multi-agent system. Instead of learning a TA for the whole system, a TA is learned for each individual agent. These individual TAs can be combined in a specific way to represent the TA for the global system. This approach can be seen as an improvement on the one presented in [16], where TAs for each agent are used to satisfy the global task specification. However, the focus of our method is on learning the TAs, rather than designing them, which removes the need for prior knowledge of the global task and reduces the cost of manual decomposition.

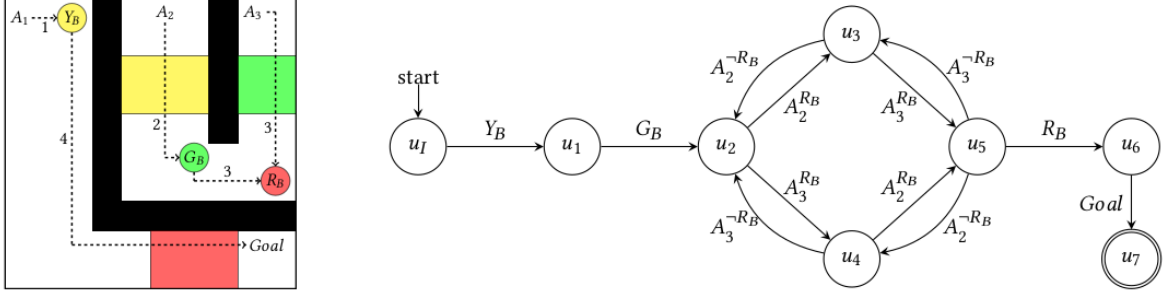


Figure 4: Environment and TA for the 3-buttons task. There are three agents A_1 , A_2 and A_3 and there are three buttons Y_B , G_B and R_B . The task and possible events is shown by the TA.

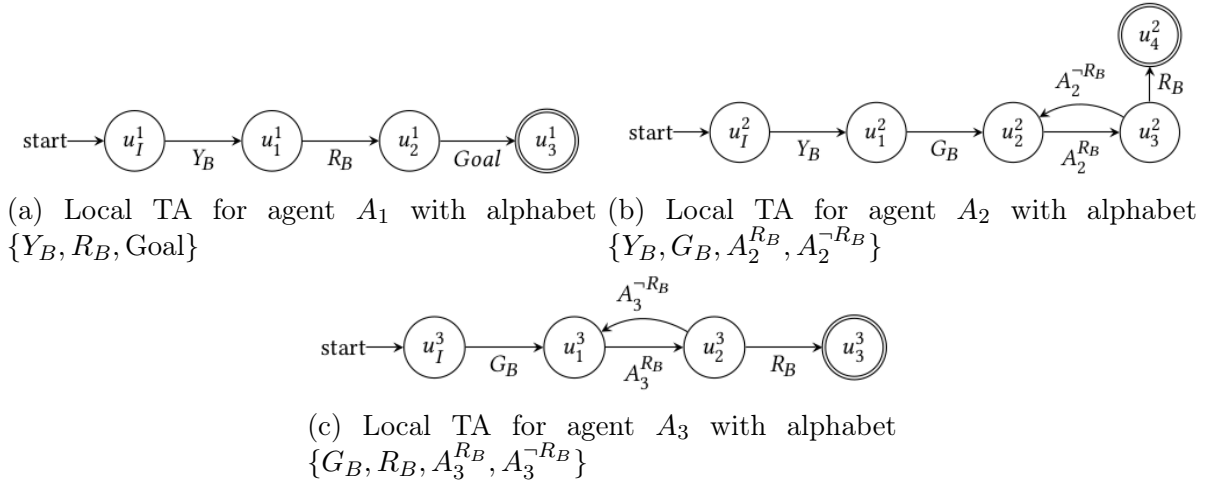


Figure 5: A set of local TAs, one for each agent given a local alphabet for each agent. This set of local TAs is equivalent to the global TA when combined and thus can be used to synthesise optimal policies.

In the approach presented in [16], a human is required to select the TAs for each agent based on a priori knowledge of the global task. For example, consider the environment and TA in Figure 4.

An example of a set of local TAs that could be chosen, such that their combination (the way to combine is via interleaving which will be discussed in Section 4.4) represents the global TA, is given in Figure 5.

In contrast, our approach leverages Algorithm 1 that can learn a TA for an agent in a single-agent setting using trajectories, traces, and reward sequences. For a multi-agent system, each agent is assigned a set of trajectories, traces, and reward sequences, from which a product MDP for the agent is learned. A TA that represents the behaviour of

the agent in the system can then be extracted from this.

By combining the TAs of all agents, the behaviour of the global system can be represented. With the proposed approach, the need for manual intervention is reduced, and learning TAs for multi-agent systems with limited prior knowledge of the proposed task becomes possible.

4.2 Local Trajectories, Traces and TAs

The first choice to be made is to decide what trajectories, traces and reward sequences we will use when learning the local TA for each agent.

In a naive approach, we can define a local TA for agent j as $\mathcal{A}_j = (\mathcal{Q}_j, q_j^0, \Sigma, \delta_j, \mathcal{F}_j)$ where $\Sigma = 2^{\mathcal{A}^{\mathcal{P}}}$. Say we have n agents, t time steps, and a state-action trajectory $((s_1^0, \dots, s_n^0), (a_1^0, \dots, a_n^0), (s_1^1, \dots, s_n^1), \dots, (a_1^t, \dots, a_n^t), (s_1^t, \dots, s_n^t))$. We can then define the local trajectory for agent i as $(s_i^0, a_i^0, s_i^1, \dots, a_i^t, s_i^t)$, which is the sequence of states and actions where we take only the component of the global state or action vector that corresponds with our chosen agent. Thereby ignoring the states and actions of the other agents. In this naive approach, the local trace and the local reward sequence are the same as the global trace and global reward sequence respectively.

However, running the TA learning pipeline on local trajectories, traces, and reward sequences would result in each agent learning the full global TA, with only a small advantage over the simplest method of modelling the entire multi-agent system as if it was a single entity. Each agent observes all the labels, so when extracting a TA from the traces, it will extract a TA that includes all the labels necessary for the completion of the global task, which is global TA. Additionally, the assumption that each agent’s local trace is identical to the global trace requires a global method of communication, which may not be feasible in situations with limited communication between agents. Thus, an alternative method is needed to address these issues.

4.3 Local Alphabets

An alternative approach when defining what traces, trajectories and reward sequences we use when learning a TA is to allow us to restrict the set of labels a given agent observes. We can then choose this set to be only the labels necessary for agent i to successfully complete its part of the task or some superset of this. So long as we can accept, for now, the assumption that we know of at least some labels that are not relevant for particular agents then this approach can reduce the size of learnt TAs.

Formally, for each agent i we define a local alphabet \mathcal{AP}_i which leads to a local set of labels $2^{\mathcal{AP}_i} = \Sigma_i \subseteq \Sigma$. We define the local trace for agent i to be $(\ell_i^0, \ell_i^1, \dots, \ell_i^n)$ where $(\ell^0, \ell^1, \dots, \ell^n)$ is the global trace and $\ell_i^j = \ell^j \cap \mathcal{AP}_i$.

We can use these local traces to learn a local TA for each agent, let's say the local TA for agent i is $\mathcal{A}_i = (\mathcal{Q}_i, q_i^0, \Sigma_i, \delta_i, \mathcal{F}_i)$.

4.4 Conditions on Local TAs Necessary for Multi-Agent Q-Learning Algorithm

Our algorithm aims to learn a set of local TAs that can be used to synthesise an optimal policy using the method proposed in [16]. The learnt TAs must satisfy the conditions for any set of local TAs to be able to be used to learn an optimal policy, the conditions were first outlined in [16]. Assuming we have learned a set of local TAs, I will describe each condition and present the theorem.

Firstly, to ensure that each agent satisfying its own local TA leads to the global task being satisfied we want our local TAs to capture the same task as the global TA when combined. We must define a way of combining local TAs, in the original work [16] the parallel composition of TAs was used. I propose a more general approach called interleaving that, unlike the original, allows for multiple events to occur at a given time step.²

²Interleaving and parallel composition are identical if each label is only a singular event, interleaving allows for labels that are sets of events. Interleaving does not impact the validity of Theorem 1 as the assumption that only one event occurs at a particular time step is not used in the original proofs in [16].

To interleave a set of TAs means each interleaved state is a product of the states of the local TAs. The initial interleaved state is the product of the initial state for each local TA. Transitions occur between two interleaved states if, for a given label l and each local TA \mathcal{A} that has a subset l' of the events in l in its label set, there was a transition labelled l' between the two local states for \mathcal{A} that are in the interleaved states. The set of accepting states in the interleaved TA is the cartesian product of the accepting states of each of the local TAs.

An illustrative example of the interleaving of two TAs is given in Figure 6.

Definition 6. An **Interleaving** of a set of TAs $\{\mathcal{A}_1, \dots, \mathcal{A}_n\}$ is defined as a TA $\mathcal{A} = (\mathcal{Q}, q^0, \Sigma, \delta, \mathcal{F})$ where:

- $\mathcal{Q} = \mathcal{Q}_1 \times \dots \times \mathcal{Q}_n$
- The initial state $q^0 = (q_1^0, \dots, q_n^0)$
- $\Sigma = 2^{\mathcal{AP}_1 \cup \dots \cup \mathcal{AP}_n}$
- Let $q^k = (q_1^{k_1}, \dots, q_n^{k_n})$ then $\delta(q^k, \sigma) = (q_1^{j_1}, \dots, q_n^{j_n})$, where $q_i^{j_i} = q_i^{k_i}$ if $(\sigma \cap \mathcal{AP}_i) \notin \Sigma_i$ and if $(\sigma \cap \mathcal{AP}_i) \in \Sigma_i$ we have that $\delta(q_i^{k_i}, \sigma \cap \mathcal{AP}_i)$ is defined, then $q_i^{j_i} = \delta(q_i^{k_i}, \sigma \cap \mathcal{AP}_i)$, and we require that $\bigcup_{i=1}^n \sigma \cap \mathcal{AP}_i = \sigma$, otherwise, $\delta(q^k, \sigma)$ is undefined
- $q^k \in \mathcal{F} \iff \forall i. q_i^k \in \mathcal{F}_i$

We desire the TA produced by interleaving to represent the global task of the entire system, this is equivalent to saying it recognises the same language as the underlying global TA. This is the case if the two TAs are bisimilar. Bisimilarity means that we can come up with some relation between the states of two TAs such that the initial states are related, and accepting states are related to all and only other accepting states. Furthermore, if a transition exists between two states in one TA, then a transition with the same label must occur between two states in the other TA, with each state in the transition related to the equivalent state in the transition in the first TA. This means the two TAs recognise the same language and have a similar structure.

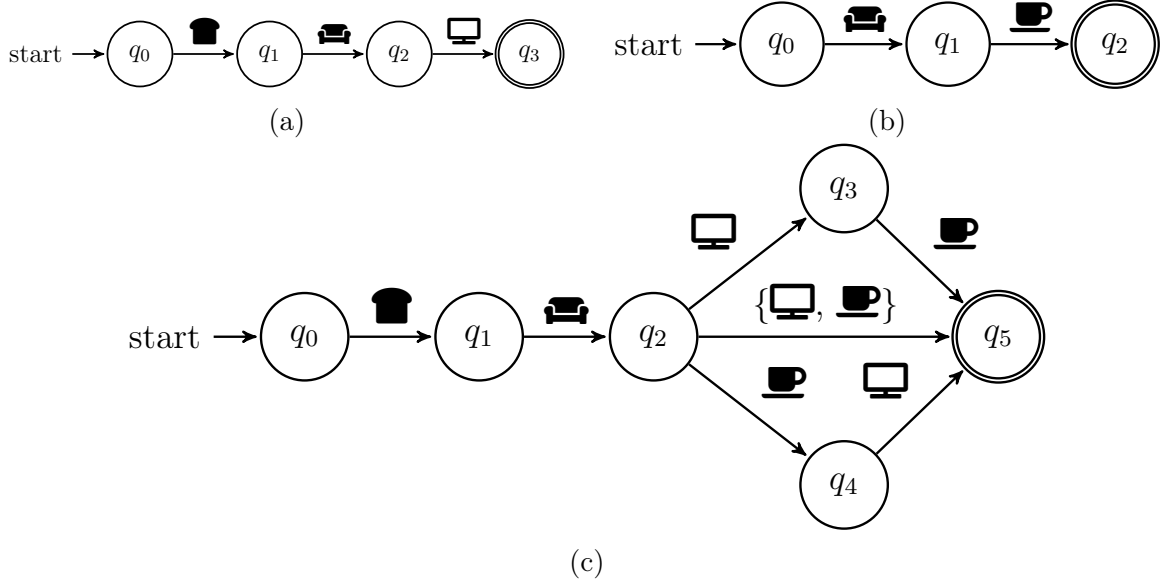


Figure 6: (a) is a TA with alphabet $\{\text{house}, \text{car}, \text{computer}\}$ and (b) is TA with alphabet $\{\text{car}, \text{coffee}\}$. (c) is the interleaving of (a) and (b)

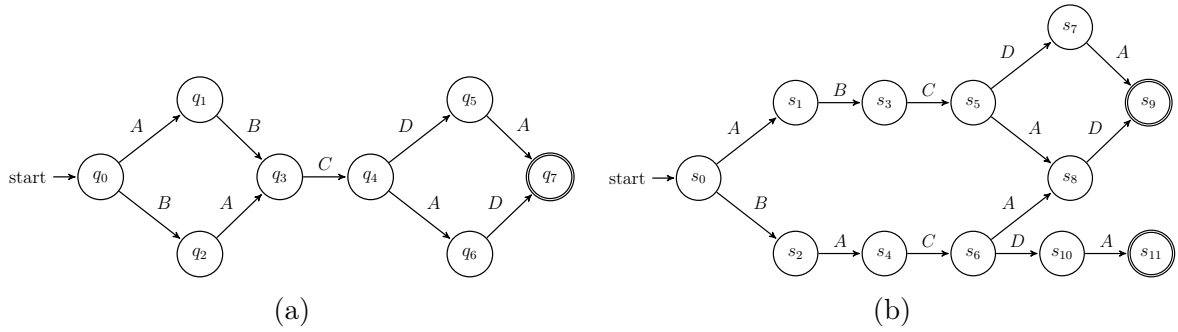


Figure 7: (a) and (b) are bisimilar under the following relation $\sim = \{(q_0, s_0), (q_1, s_1), (q_2, s_2), (q_3, s_3), (q_3, s_4), (q_4, s_5), (q_4, s_6), (q_5, s_7), (q_5, s_{10}), (q_6, s_8), (q_7, s_9), (q_7, s_{11})\}$

We can see a simple example of two bisimilar TAs in Figure 7, which demonstrates how two TAs of differing size and structure can represent the same language. It also illustrates how the learnt TAs may not necessarily interleave to a minimal TA describing the task, but this is not a concern as we never directly use the interleaved TA we just require it to meet certain conditions. The formal definition of bisimilar is as follows.

Definition 7. Two TAs $\mathcal{A}_1 = (Q_1, q_1^0, \Sigma, \delta_1, \mathcal{F}_1)$ and $\mathcal{A}_2 = (Q_2, q_2^0, \Sigma, \delta_2, \mathcal{F}_2)$ are **bisimilar** denoted $\mathcal{A}_1 \cong \mathcal{A}_2$ if there exists a relation $\sim \subseteq Q_1 \times Q_2$ such that :

- $q_1^0 \sim q_2^0$
- For every $q_1 \sim q_2$, $q_1 \in \mathcal{F}_1$ if and only if $q_2 \in \mathcal{F}_2$

- If $\delta_1(q_1, \sigma) = q'_1$ for some $\sigma \in \Sigma$, then there exists $q'_2 \in \mathcal{Q}_2$ such that $\delta_2(q_2, \sigma) = q'_2$ and $q'_1 \sim q'_2$
- If $\delta_2(q_2, \sigma) = q'_2$ for some $\sigma \in \Sigma$, then there exists $q'_1 \in \mathcal{Q}_1$ such that $\delta_1(q_1, \sigma) = q'_1$ and $q'_1 \sim q'_2$

We also require that all the agents agree on the state of the higher-level features of the environment at any given time. This can be captured by ensuring our labelling function is decomposable.

Definition 8. A labelling function $L : \mathcal{S} \rightarrow \Sigma$ is considered **decomposable** with respect to local alphabets $\mathcal{AP}_1, \mathcal{AP}_2, \dots, \mathcal{AP}_n$, and local label sets $\Sigma_i = 2^{\mathcal{AP}_i}$, if there exists a collection of local labelling functions L_1, L_2, \dots, L_n with $L_i : S_i \rightarrow \Sigma_i$ such that $L(s)$ outputs event $e \in \mathcal{AP}$ if and only if $L_i(s_i)$ outputs label e for every L_i such that e belongs to the local alphabet of agent i .

Finally, it must be the case that all labels necessary for the completion of the task are attainable when each agent has been given its restricted alphabet. That property is captured by ensuring each label is in the local label set of each of its agents from some controlling set. As outlined in the following definition.

Definition 9. A label $l \in \Sigma$ is **controlled** by a set of agents $C_l \subseteq I$ if for all combinations of local states of agents not in C_l , $\{q_{jk} | \forall k \in I \setminus C_l. q_{jk} \in Q_k\}$ there exists some set of states for the agents in C_l , $\{q_{ik} | \forall k \in C_l. q_{ik} \in Q_k\}$ such that $L(q) = l$ where $q = (q_1, q_2, \dots, q_n)$, $q_k = q_{ik}$ if $k \in I$ and $q_k = q_{jk}$ otherwise.

Given the informal description of the conditions for this theorem I will now state it formally, full justification is given in [16].

Theorem 1. *Given a labelled Markov Game $\mathcal{G} = (\mathcal{I}, \mathcal{S}, \mathcal{A}, P, R, \gamma, \mathcal{AP}, L)$, a global TA $\mathcal{A} = (\mathcal{Q}, q^0, \Sigma, \delta, \mathcal{F})$ that encodes the task for \mathcal{G} , and a set of local TAs $\{\mathcal{A}_1, \dots, \mathcal{A}_n\}$ where $\mathcal{A}_i = (\mathcal{Q}, q_i^0, \Sigma_i, \delta_i, \mathcal{F}_i)$. An optimal policy for the labelled Markov Game \mathcal{G} can be learnt using just the local TAs if:*

- *The Interleaving (see Definition 6) of the set of local TAs $\{\mathcal{A}_1, \dots, \mathcal{A}_n\}$ is bisimilar (see Definition 7) to the global TA \mathcal{A}*

- The labelling function L is considered decomposable with respect to the local alphabets $\{\Sigma_1, \dots, \Sigma_n\}$ (see Definition 8)
- For each label l that is necessary for the completion of the task, $l \in \Sigma_i$ for all the agents $i \in C$, in at least one of l 's controlling sets C (see Definition 9)

4.5 The Learnt TAs

In this section, I propose a way in which we can learn a set of TAs that meet the conditions in Theorem 1.

Proposition 1. *Given a labelled Markov Game \mathcal{G} and a global TA \mathcal{A} (or a modification on this global TA discussed in Section 4.6). Assume there exists a decomposition into a set of local TAs such that the decomposition satisfies the conditions in Theorem 1. Then if we assign to each agent i a local alphabet $\tilde{\mathcal{A}}\mathcal{P}_i$, we can get the local traces for each agent as defined in Section 4.3, and the local trajectories and reward sequences as defined in Section 4.2. Then if we use these in conjunction with the TA learning algorithm outlined in Algorithm 1, we can learn a set of local TAs that together meet the conditions in Theorem 1, therefore can be used to learn an optimal policy using the algorithm in [16].*

Proof. For agent i , given a local alphabet $\mathcal{A}\mathcal{P}$, and sets of local traces, trajectories, and reward sequences we can use Algorithm 1 to learn a TA $\overline{\mathcal{A}}_i$ for the task encoded by \mathcal{A}_i . Ideally, we would want $L(\mathcal{A}_i) = L(\overline{\mathcal{A}}_i)$, however, this is not always the case as discussed in Section 3.3. How to handle the case where $L(\mathcal{A}_i) \neq L(\overline{\mathcal{A}}_i)$ is discussed in Section 4.6.

Assuming we have, if necessary, undertaken the steps in Section 4.6, we can assume each of our learnt TAs $\tilde{\mathcal{A}}_i$ accepts the same language as \mathcal{A}_i . Then by Lemma 1 the interleaving of $\{\tilde{\mathcal{A}}_1, \dots, \tilde{\mathcal{A}}_n\}$ is bisimilar to the global TA.

We have already assumed that the labelling function L is decomposable with respect to the local alphabets that we have used for the learnt TAs. Therefore this condition is met.

Finally, we also assumed that each label belonged to all the local label sets of the agents in at least one of its controlling sets. So this condition is also met.

Therefore the learnt TAs meet all the conditions in 1. \square

As part of the proof we need to show that the interleaving of the learnt local TAs, assuming they have the same languages as the underlying local TAs, is guaranteed to be bisimilar to the global TA, which is shown with the lemma below.

Lemma 1. *If a set of TAs $\{\tilde{\mathcal{A}}_1, \dots, \tilde{\mathcal{A}}_n\}$ each have the same language as another set of TAs $\{\mathcal{A}_1, \dots, \mathcal{A}_n\}$, meaning for all i we have $L(\tilde{\mathcal{A}}_i) = L(\mathcal{A}_i)$, and the interleaving of set of TAs $\{\mathcal{A}_1, \dots, \mathcal{A}_n\}$ is bisimilar to some TA \mathcal{A} . Then the interleaving of the set of TAs $\{\tilde{\mathcal{A}}_1, \dots, \tilde{\mathcal{A}}_n\}$ is also bisimilar to the TA \mathcal{A} .*

Proof. Let $\tilde{\mathcal{A}}$ be the interleaving of the set $\{\tilde{\mathcal{A}}_1, \dots, \tilde{\mathcal{A}}_n\}$, and let \mathcal{A}' be the interleaving of the set $\{\mathcal{A}_1, \dots, \mathcal{A}_n\}$.

Given TAs are DFAs, two TAs have the same language if and only if they are bisimilar to each other. Thus $L(\mathcal{A}) = L(\mathcal{A}')$ as \mathcal{A} is bisimilar to \mathcal{A}' . If we can show $L(\mathcal{A}') = L(\tilde{\mathcal{A}})$, then it follows that $L(\mathcal{A}) = L(\mathcal{A}') = L(\tilde{\mathcal{A}})$, and then we could say \mathcal{A} is bisimilar to $\tilde{\mathcal{A}}$.

It is clear from the definition of interleaving that for any TA \mathcal{T} , where \mathcal{T} is the interleaving of a set of TAs $\{\mathcal{T}_1, \dots, \mathcal{T}_n\}$ and a sequence of labels $\ell^1 \ell^2 \dots \ell^m$, that $\ell^1 \ell^2 \dots \ell^m \in L(\mathcal{T})$ if and only if for all i , we have that $\ell_i^1 \ell_i^2 \dots \ell_i^m \in L(\mathcal{T}_i)$, where $\ell_i^j = \ell^j \cap \mathcal{AP}_i$, and for all j we have that $\bigcup_{i=1}^n \ell_i^j = \ell^j$.

Thus the accepting language of an interleaved TA depends only on the languages of the set TAs in the set used to create it. Thus if the languages $L(\mathcal{A}_i) = L(\tilde{\mathcal{A}}_i)$ for all i , then $L(\mathcal{A}') = L(\tilde{\mathcal{A}})$.

Therefore, $\tilde{\mathcal{A}}$ is bisimilar to \mathcal{A} . \square

It is clear a wide variety of tasks can be decomposed in this way, for example, if an agent has an independent subtask within the global task this can be captured with appropriate local alphabets. Similarly, sequences of tasks with different relevant label sets could be decomposed like this. It is also clear that if a task involves synchronisation by a particular set of agents at a given label it is easy to capture this by assuring that this label is in the

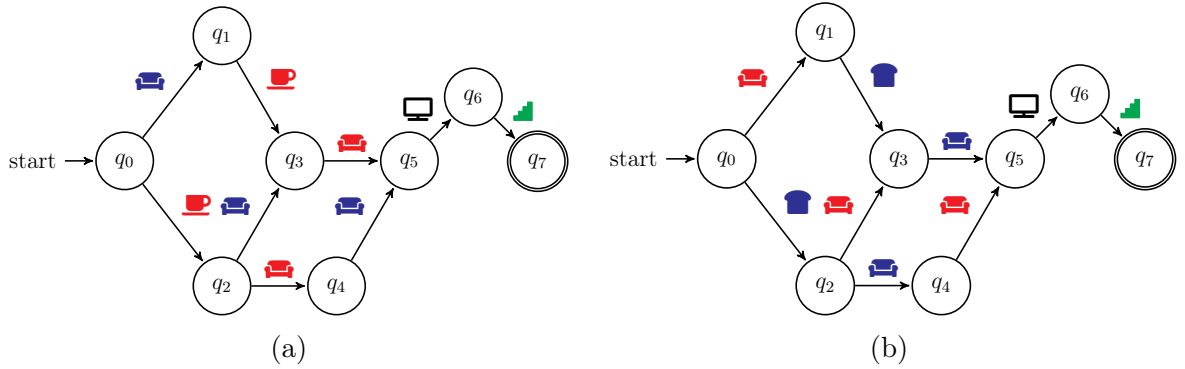


Figure 8: (a) An 8-state local TA for Red and (b) An 8-state local TA for Blue.

alphabets of those agents. In fact, any task can be decomposed like this if we give each agent the global alphabet as its local alphabet, but clearly in this case each agent would just learn the global TA. Thus there is no benefit in tasks where every agent needs to observe all labels in order for the global task to be complete. Example 3 demonstrates a simple example where there the local TAs are significantly smaller than the global TAs.

Example 3. *It is evident from figure 2b that a simple Markov Game can have a very large TA, we can imagine a local TA for each agent as follows, say that agent Red has a TA limited to the red labels, black labels, green labels and the blue couch. Similarly Blue has a TA limited to blue labels, black labels, green labels and the red couch. Then we would get the local TA in figures 8a and 8.*

4.6 Attainable TA

As discussed in Proposition 1 and outlined in Section 3.3, one of the potential issues with using Algorithm 1 to learn the TAs is that it can be the case that extra labels can appear in the learnt TA, that are not necessarily in the underlying TA. This happens when a label l is necessary for some part of the TA, and then in another part of the TA, there is another label that only appears in traces after the label l . It could be that this l is not necessary for the completion of the task and thus is not present in the underlying TA, but in the learnt TA this l transition will always be present as it appears in that position in all traces that receive a reward. When the l is not present in the underlying TA at all then we can remove it from the learnt TA using post-processing steps as discussed in

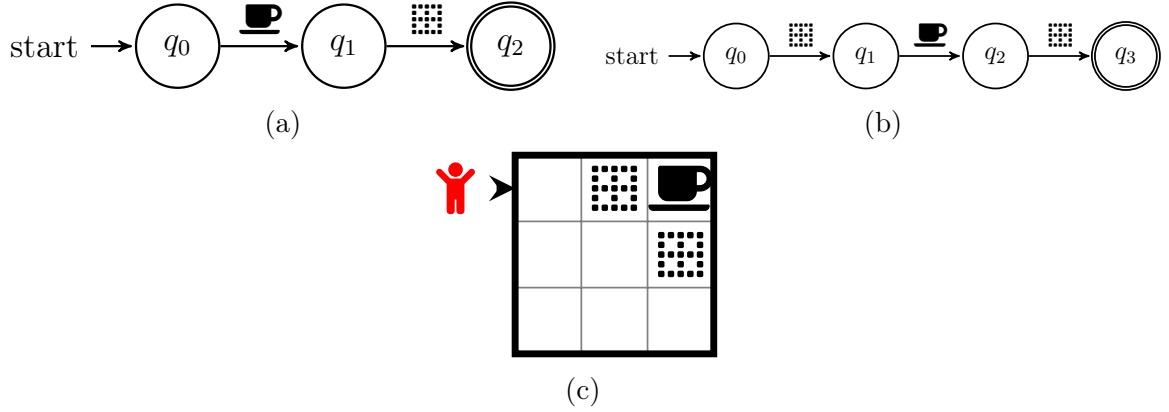








Figure 9: (c) is a Grid world where  is only accessible by first encountering . (a) Is the true underlying TA whilst (b) is the learnt TA due to the fact that  always appears directly before  in any trace, even though it isn't in the underlying TA

3.3, but if it is present in some other part of the TA then this isn't possible. A simple example of an environment and task where this is the case can be seen in Figure 9.

This means that our learnt TAs won't necessarily have the same languages as the underlying TAs, however, this doesn't stop us from learning an optimal policy. First, consider the following definition.

Definition 10. A trace, $\ell_1\ell_2\dots\ell_n$, is considered **Markov Game-attainable** (or similarly **MDP-attainable**) for a labelled Markov Game (or labelled MDP), \mathcal{G} , if there is some state-action trajectory, $(s_1, a_1, s_2, a_2, \dots, a_{n-1}, s_n)$, such that for all i , $L(s_i) = \ell_i$ and $P(s_i|a_{i-1}, s_{i-1}) > 0$. Define the function $\alpha(\mathcal{G}, \ell_1\ell_2\dots\ell_n)$ as follows:

$$\alpha(\mathcal{G}, \ell_1\ell_2\dots\ell_n) = \begin{cases} 1 & \text{if } \ell_1\ell_2\dots\ell_n \text{ is Markov Game-attainable with respect to } \mathcal{G} \\ 0 & \text{otherwise} \end{cases}$$

In the example in Figure 9, all attainable traces have  before they have . So $(\emptyset, \emptyset, \text{cup})$ is not MDP-attainable whereas $(\emptyset, \text{grid}, \text{cup})$ is MDP-attainable.

Therefore we can define the following TA, based on the idea of attainable traces:

Definition 11. Given a TA, \mathcal{A} , its language, $L(\mathcal{A})$, and a labelled Markov Game (or MDP), \mathcal{G} the **attainable-TA**, \mathcal{A}^α , is a TA such that $t \in L(\mathcal{A}^\alpha)$ if and only if $t \in L(\mathcal{A})$

and $\alpha(\mathcal{G}, t) = 1$.

3

The global attainable-TA and the local attainable-TAs are the TA that we actually learn by applying the algorithm for TA learning. For the example in Figure 9a the local attainable-TA is in fact the TA in Figure 9b. Using these TAs for optimal policy learning instead of the actual underlying TA does not have an impact as the same attainable traces are accepted in both TAs. Therefore it is acceptable in the case that the attainable-TA is different to the true underlying TA to instead aim to learn a set of local TAs where their interleaving is bisimilar to the attainable-TA.

By using the attainable-TA we avoid the case where the TA-Learning algorithm for local TAs does not learn a TA with the correct language, thus we can be certain our method works for these examples.

4.7 Decomposable Labelling function

In Definition 8 for decomposable, it says that local labelling functions L_i must exist. This is a necessary condition for the successful use of the Q-Learning algorithm in [16]. Also, this is necessary for the TA learning algorithm we employ. As it learns a product MDP which relies on the labelling function being an injective mapping from states to labels.

There are many examples where the local alphabets don't lead to properly defined local labelling functions. For example, say we have agents 1 and 2, we could have the trajectory $((s_1^0, s_2^0), (a_1^0, a_2^0), (s_1^0, s_2^1))$ and trace (A, B) . Then if we restrict to the local case with local label set $\{A, B\}$ for agent 1, we get a local trajectory (s_1^0, a_1^0, s_1^0) and a trace (A, B) . We can see from this that in the local case, agent 1 doesn't change state but has a different label at each time step. Therefore we cannot define a local labelling function. This is illustrated in Figure 10.

³In general the subset of a regular language is not always regular, but for the tasks we are considering this is always true. Hence why we can define a TA for the attainable language.

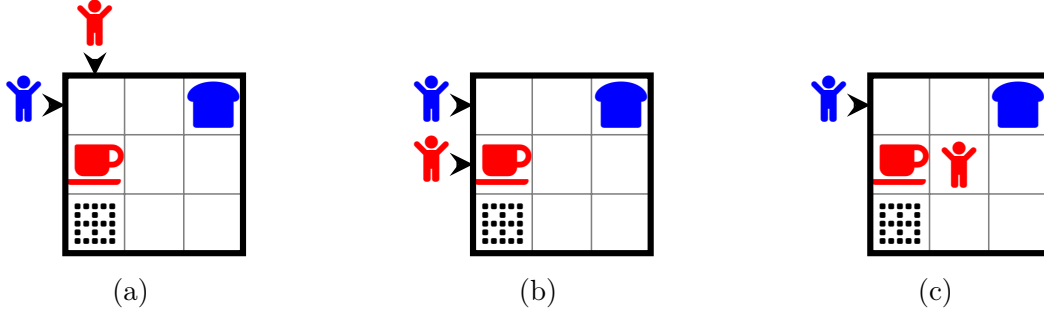


Figure 10: Consider the local traces for a local alphabet of $\{\text{red mug}, \text{blue shirt}\}$ with this series of positions for Red and Blue. It illustrates the problem outlined in this section

Many simple examples lead to impossible local labelling functions. However, there is a systematic way to overcome this.

1. Firstly identify all the labels, $\{\ell_1, \ell_2, \dots, \ell_m\}$, that agent i sees when it is in state s_i^j at different time steps.
2. For each $\ell_k \in \{\ell_1, \dots, \ell_n\}$, create a new state $s_i^{j\ell_k}$.
3. Then in the Markov Game, for all states where $L((s_1, \dots, s_i^j, \dots, s_n)) \cap \mathcal{AP}_i = \ell_k$ replace s_i^j with $s_i^{j\ell_k}$. Do this for each label $\ell_k \in \{\ell_1, \dots, \ell_n\}$.
4. Repeat the above 3 steps for every state for every agent.

If we consider this approach applied to the example we used to illustrate the problem then we get the following trajectory: $((s_1^{0A}, s_2^0), (a_1^0, a_2^0), (s_1^{0B}, s_2^1))$ and the trace is still (A, B) . If we once again consider $\{A, B\}$ as the local label set for agent 1, we get the local trajectory $(s_1^{0A}, a_1^0, s_1^{0B})$ and local trace (A, B) . Clearly, the local labelling function is now well defined as we have different states at each time step and different labels.

In practice, the only labels that need to be considered are the labels that appear in more than one alphabet. We can identify the labels by exploring the environment and recording all the observed labels for each state and applying the above method where there is more than one for a given local state. This does increase the size of the MDPs we are trying to learn. As we have an extra state for each occurrence of this problem.

4.8 Agent Communication

When designing multi-agent systems, considering the communication protocol used by the agents is crucial. Algorithm 2 requires certain assumptions to be met for successful learning of local TAs. These assumptions include that each agent is only aware of its own MDP states and only observes labels that are in its own local label set.

We can split the local label set into two sets. One is the labels that result from a given agent's actions, meaning the agent belongs to one of the label's controlling sets. These labels are directly observed by the agent and no communication is required to see these. The other is the labels that belong to the local label set of an agent but are not controlled by the agent. The agent won't directly observe these thus some communication is required in order for the agent to observe them.

To enable all agents to observe the labels in their local label sets that they don't control, we must assume the following communication occurs:

1. For each agent i , consider its local alphabet \mathcal{AP}_i .
2. For each other agent j , take the intersection $\mathcal{AP}_i \cap \mathcal{AP}_j$.
3. Whenever agent i observes an event $e \in \mathcal{AP}_i \cap \mathcal{AP}_j$ it then communicates that this event has occurred to agent j .

Any communication protocol that respects the above assumptions would allow Algorithm 2 to succeed. For instance, if there is a global controller, each agent can report all their observations to this controller, which can then send messages to the relevant agents when an event occurs that is in their respective alphabets. Optimal communication protocols can be designed for different examples so long as they meet the conditions above.

4.9 Multi-Agent TA Learning Algorithm

Taking the idea of using local alphabets, an algorithm for learning local TAs for a labelled Markov Game is presented in algorithm 2.

Algorithm 2 Learning a set of local TAs in an unknown labelled Markov Game

Input: put as set of agents into an (unknown) labelled Markov Game \mathcal{G}

Output: A set of TAs $\{\mathcal{A}_i^* | i \in \mathcal{I}\}$ where \mathcal{I} is the set of agents, such that their interleaving represents a task specification

- 1: OBSSEQ \leftarrow collect episodes of corresponding trajectories, traces, and reward sequences from the Markov Game.
 - 2: LOCALOBSSEQ \leftarrow using the local alphabet for each agent process OBSSEQ to create the local trajectories, traces and reward sequences for each agent.
 - 3: **for** Agent $i \in \mathcal{I}$ **do**
 - 4: **function** LEARNPRODUCTMDP(OBSSEQ) ▷ Step 1
 - 5: Use a HMM/POMDP learning algorithm with LOCALOBSSEQ to learn and return an estimate of the transition probability distribution \hat{P}_π of the spatial MC $\hat{\mathcal{M}}_\pi$.
 - 6: Use a HMM/POMDP learning algorithm initialised with \hat{P} as an inductive bias and trained with LOCALOBSSEQ to learn and return an estimate of the transition probability distribution \hat{P}^\otimes of the product MDP $\hat{\mathcal{M}} \otimes \hat{\mathcal{A}}$.
 - 7: **function** DISTILTA($\hat{\mathcal{M}} \otimes \hat{\mathcal{A}}$) ▷ Step 2
 - 8: Determinise $\hat{\mathcal{M}} \otimes \hat{\mathcal{A}}$ using Cone Lumping method to return the MDP-restricted TA $\hat{\mathcal{A}}_{\hat{\mathcal{M}}}$
 - 9: **function** POSTPROCESS($\hat{\mathcal{A}}_{\hat{\mathcal{M}}}$) ▷ Step 3
 - 10: Remove environmental bias and minimise $\hat{\mathcal{A}}_{\hat{\mathcal{M}}}$
-

4.10 Local Reward Functions

Another approach we can take, in addition to the above, is to consider an alternative definition of reward function where rather than having a reward that is identical for all agents, we define $R_i : (\mathcal{S} \times \mathcal{A})^+ \times \mathcal{S} \rightarrow \mathbb{R}$ as the local reward function for each agent i . Then the overall goal is to maximise the expected sum of all the local rewards, $E_\pi[\sum_{t=0}^{\infty} \gamma^t \sum_{i=1}^n r_i^t]$. With this approach, we could then assume our reward function is designed such that each agent gets a reward of 1 when it has completed its part of the task. Then when we attempt to learn the local TA using this local reward function we will only learn a TA with labels that appear before achieving the reward of this agent and should ignore any labels that appear after.

It is clear that we will only learn smaller TAs in cases where at least some agents have local tasks that they get a reward before the final agent is rewarded, if all agents get a reward at the same time then all agents will have the same TA as in the case where we do not have local reward functions.

This kind of task is still compatible with the method in [16] thus learnt TAs will still be able to be used to learn an optimal policy. Additionally, the only changes required to Algorithm 2 is that each agent now receives its local reward sequence rather than the global one.

5 Experimental Results

In this project, I have proposed a novel approach for learning both TAs and MDPs for individual agents in a multi-agent system. To evaluate the effectiveness of this approach, I will conduct a series of experiments on tasks with varying degrees of complexity, agent numbers, and grid sizes.

To the best of my knowledge, this is the first attempt to simultaneously learn TAs and MDPs for individual agents in a multi-agent system. While there is one existing approach [4] for learning TAs in multi-agent systems, I was unable to obtain the code, making it difficult to benchmark my work against existing methods. As a result, I will provide benchmarks for a range of tasks to demonstrate the effectiveness of the algorithm, compare it to the naive approach of learning a global TA, and discuss insights gleaned from the experiments.

Specifically, I will examine the scalability of the algorithm, the effectiveness of learning TAs with low-probability events, the potential advantages of this method with regard to transfer learning and the impact of using local reward functions rather than global ones. By exploring these topics, I hope to shed light on the strengths and weaknesses of the proposed approach and provide a foundation for future research in this field.

5.1 Setup

The environments for the main examples we will consider are the grids in Figure 11. Initially, for each grid, we will attempt to learn TAs for the two global tasks in Figure 12. The grid worlds all share the same set of events $\mathcal{AP} = \{\text{red}, \text{blue}, \text{green}, \text{grid}\}$, which have been distributed at random across the grid.

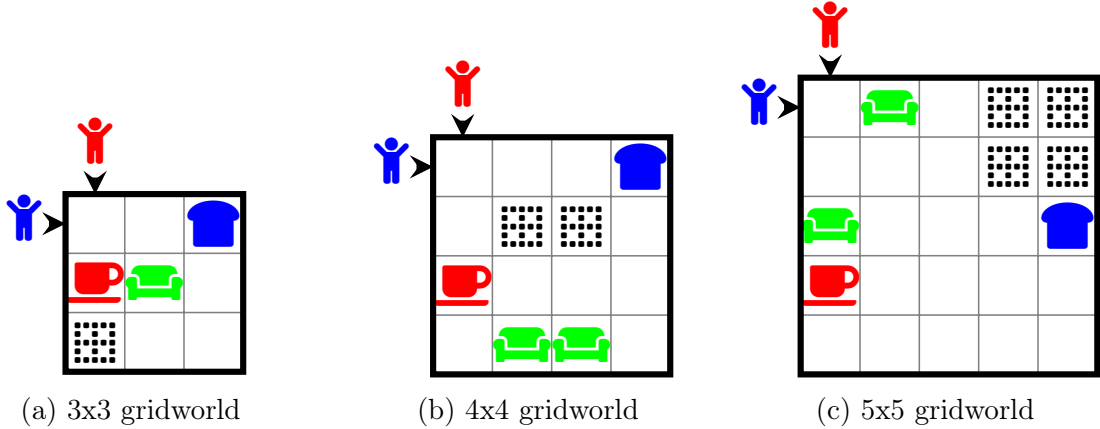


Figure 11: Illustrations of the grid worlds used in the experiments. The agents always start in the top left-hand corner of the grid. Blue (or Red) labels only occur when the Blue (or Red) agent is at that cell. Green labels occur when both agents are in the cell. Black labels occur when any agent is in the cell

For each experiment, the agents first collect a set of traces, trajectories and reward sequences by exploring the grid uniformly at random. Then step 2, which can be completely parallelised, is to learn the product MDP $\mathcal{M} \otimes \mathcal{A}$ for each agent. These two steps were implemented in C++, and the rest of the pipeline is implemented in Python. Steps 3 and 4 extract the DFA underlying the product MDP and remove any environmental bias, these two steps combined take less than 1s for all the experiments that were undertaken and thus will not be considered any further in the analysis of the running time. Instead, this analysis will focus on the runtime of the learning of the product MDPs. Specifically, we will measure the run time in terms of the maximum run time out of the n agents due to the fact that the learning of each of the agent’s product MDPs can be done in parallel. All experiments are an average of 3 runs and were carried out on an Intel i7-8565U CPU (1.8 GHz x 8).

5.2 Performance on Tasks and Learnt TAs

Both the proposed method and the naive method were run on the Tasks outlined in Figure 12 for each grid in 11. For the runs of Algorithm 2 Agent 1 (Red) was given the alphabet $\mathcal{AP} = \{\text{red cup}, \text{green sofa}, \text{black grid}\}$ and Agent 2 (Blue) was given the alphabet $\mathcal{AP} = \{\text{blue shirt}, \text{green sofa}, \text{black grid}\}$. Thus the assumed knowledge of this task a priori is only that Agent 1 is not interested in

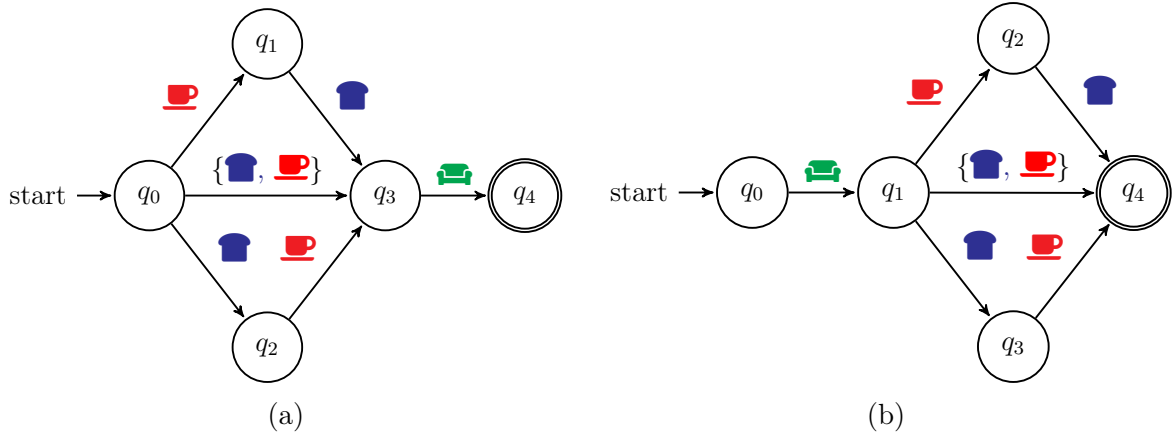


Figure 12: Two different TAs for the environments in Figure 11

. $\{\text{blue shirt}, \text{red cup}\}$ stands for when we get both events at the same time. Any event that does not appear as a transition can be considered a self-loop.

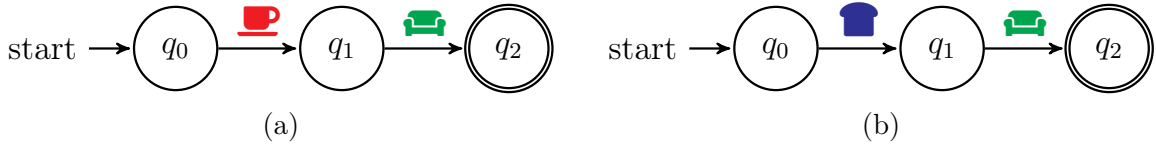


Figure 13: The learnt TAs for agents 1 and 2 for the Task in Figure 12a.

blue shirt and that Agent 2 is not interested in red cup. For the naive approach, which is a direct application of Algorithm 1 where the product of the agents' states is used as the states in the MDP, there is no a priori knowledge of the task.

Algorithm 2 learnt correct local TAs for both tasks for all 3 grids, the exact run times are presented in Table 1. The learnt TAs for Task 12a are presented in Figure 13 and the learnt TAs for Task 12b are presented in Figure 14.

The naive approach timed out on all 3 grid sizes for both examples, specifically, it was unable to converge within 3 hours. This provides clear evidence for the utility of my proposed algorithm, as it is able to learn TAs for tasks that current methods fail to converge for.

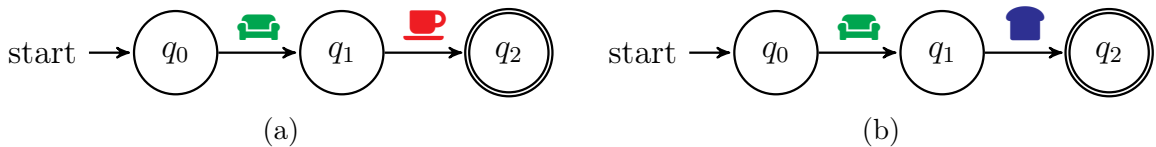


Figure 14: The learnt TAs for agents 1 and 2 for the Task in Figure 12b.

In these examples we have learnt 3 state TAs for each agent, however, the run times are comparable with those for learning 4 state TAs using Algorithm 1 in the single agent setting. This is because even though we are learning a 3-state TA for each agent, the DFA underlying the product MDP is, in fact, a 4-state DFA that is then simplified to a 3-state TA in post-processing. This is necessary for the examples we have considered because of the kind of traces that we observe. Consider the global trace $(\emptyset, \text{🚗}, \emptyset, \emptyset, \text{🚗}, \emptyset, \text{🚗})$ for the task in Figure 12b. Then the local trace for agent 1 is as follows $(\emptyset, \text{🚗}, \emptyset, \emptyset, \text{🚗}, \emptyset, \emptyset)$ but the reward sequence is $(0, 0, 0, 0, 0, 0, 1)$. This means that Agent 1 doesn't get a reward immediately after seeing 🚗 , instead it receives a reward after being in some other state with no label. Thus the DFA underlying the product MDP will include a transition into the accepting state from some empty label. This then does not allow us to extract the correct TA.

We can alleviate this issue by introducing an extra state into the TA underlying the product MDP. This extra state effectively represents a state where we are waiting for the other agent to do its part of the task. We transition into this state after seeing 🚗 , and then can transition out of it into the accepting state after moving to and from any combination of MDP states. Then when extracting the TA from the NFA it is easy to identify this state as meaningless to the task of Agent 1 as we can move out of it into the accepting state at any time. Thus we can reduce it to the 3-state TA we would like. Similar reasoning is why Agent 2's underlying TA is also of size 4 rather than 3.

This need to have a larger underlying TA than our TA is only a runtime issue, the existing algorithm for learning TAs already has to guess the size of the underlying TA until it successfully extracts a TA that describes the language of accepting traces. So no alterations to the algorithm need to be made to account for this case.

5.3 Scalability

As is clear from the results in Table 1 the time taken to learn the local TAs can increase exponentially with the number of MDP states for each agent. Unfortunately, this is due


Table 1: Parameters and results for Experiment 1, where the maximum time for convergence out of the two agents for the Baum-Welch algorithm over three runs was measured for varying grid-world and TAs.

Grid size	Task	Episode Length	No. Episodes	Convergence Time (s)
3×3	a	70	275	451.3
3×3	b	70	275	432.3
4×4	a	100	1000	3532.7
4×4	b	100	1000	3486.7
5×5	a	130	2000	17232.7
5×5	b	130	2000	17856.3

Table 2: Parameters and results for Experiment 2, where the maximum time for convergence out of all agents for the Baum-Welch algorithm over three runs was measured for varying numbers of agents.

No. Agents	Episode Length	No. Episodes	Convergence Time (s)
3	200	450	821.3
4	250	450	1465.7
5	300	450	2112.7

to the underlying method for learning the TAs [1] and is common to other TA learning methods including SAT-based and ILP-based approaches [1, 4]. It was demonstrated in [1] that the time taken to learn a TA can also grow exponentially with the size of the TA.

However, one of the benefits of this multi-agent approach is that TAs for complex tasks with a large number of agents can still be efficiently learnt so long as the local TAs are small enough. I will demonstrate that with a range of examples called the rendezvous task. The rendezvous task is an extension of the task in 12b, where each agent must first synchronise at some state, and then proceed to its own goal state. Grids are presented in Figure 15. I attempted to learn 3, 4, and 5-agent rendezvous, which have global TAs of size 17, 33 and 65 respectively and using existing methods would be completely intractable. The alphabets for each agent were  and their goal state label. Learning local TAs instead makes these problems tractable, as we learn local TAs of size 4, run times are presented in Table 2.

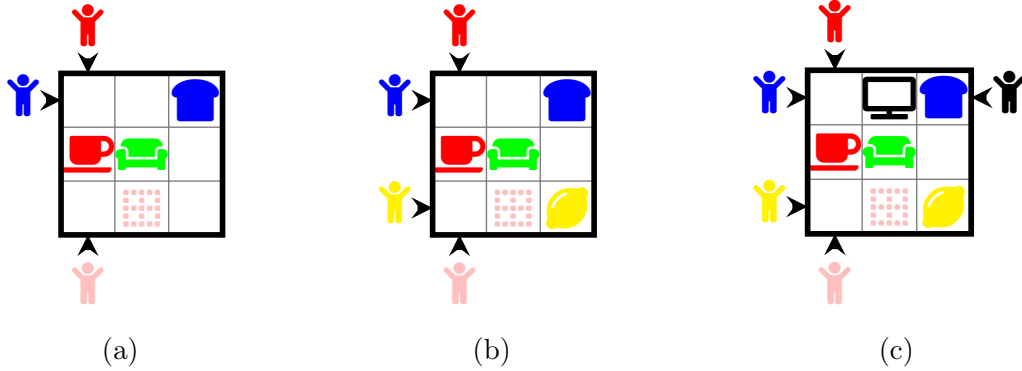




Figure 15: Illustrations of the grid worlds used in the experiments for rendezvous of more than 2 agents.

5.4 Events with Low Probability

When learning TAs in multi-agent environments, some events/labels can appear very infrequently in traces. An example is the  in the 3x3 environment in Figure 11. This label only appears when both agents are on the centre cell. Assuming the agents are distributed randomly, the probability of  occurring at any given timestep is $1/81$. As the number of agents increases, rare events have even lower probabilities. There are multiple issues that are caused by this.

First, the scarcity of certain events means necessitates longer or more exploration sequences to ensure enough of the training sequences receive rewards. This leads to longer run times learning the TA, however, this issue is not specific to my method and would apply to any technique attempting to learn TAs from traces of multi-agent environments.

Secondly, the underlying MDP for each agent contains some transitions with very small probability, for example transitioning to a state where both agents occupy a specific cell. This kind of state can be present in our MDP when we extend the environment as discussed in Section 4.7. Consequently, these small probabilities also appear in the learnt product MDP.

Even when the Baum-Welch algorithm for learning the product MDP converges there are still some small erroneous probabilities scattered through the transition matrix. This is an inherent limitation of using HMM learning as our means of learning the underlying

TA. To address this, in both Algorithm 1 and 2 we ignore values less than some ϵ when extracting the NFA from the product MDP. This leads to a problem if we have some important transition probabilities in the product MDP that are also less than ϵ purely because they are low-probability events. In which case we would not learn the correct TA.

In more complex tasks involving a large number of agents or a larger grid, it may become impossible to select a ϵ that avoids this issue. To overcome this challenge, I propose including active learning[2]. Specifically, the exploration policy would be updated based on what traces lead to rewards in order to make important low-probability events more frequent in the traces. Although this approach may result in learned MDPs that do not accurately represent the agent’s dynamics, it is not a significant concern since the primary goal is to extract correct TAs for optimal policy learning.

5.5 Local Reward Functions

All the previous experiments have focused on the case where the agents all receive a reward at the same time when the task is completed. Another way to model the system is to assume the task is such that agents receive a reward when they complete their part of the task, as discussed in 4.10.

It is clear this approach will lead to no improvements for the task in Figure 12a, so I have only run experiments for the task in Figure 12b. The results are presented in Table 3. The same local TAs were learnt as when we didn’t have local reward functions, but the run time has significantly decreased. This was due to each agent getting the reward immediately after the last relevant label, which meant the underlying NFA for the product MDP was only of size 3.

The results show that utilising local reward functions for tasks where they exist can lead to a significant improvement in the runtime. In addition, they can decrease the size of learnt TAs, specifically where some agent sees a label after it has received its local reward but before the global task is completed. This label could be present in the TA if the global

Table 3: Parameters and results for Experiment 3, where the maximum time for convergence out of the two agents for the Baum-Welch algorithm over three runs was measured for varying grid-world and TAs.

Grid size	Task	Episode Length	No. Episodes	Convergence Time (s)
3×3	b	70	275	45.7
4×4	b	100	450	348.3
5×5	b	200	5000	3643.7

reward signal was used but if we can assume knowledge of some local reward signals then this label would be ignored. Hence reducing the size of the TA. Both these improvements combined show that the proposed method has the potential for stronger performance if we can assume knowledge of local reward functions.

6 Conclusion

In this project, I introduced an algorithm that successfully learns local TAs in multi-agent RL systems with unknown environments and sparse, non-Markovian reward signals. I have ensured that these learnt TAs can be used to efficiently synthesise optimal policies with the algorithm proposed in [16]. My algorithm was the first (alongside the algorithm proposed in [4]) to provide a procedure specifically for this problem and exceeds the capabilities of the existing naive methods. The main contribution is restricting each agent to a local set of traces which allows it to learn a local TA that is smaller than the global TA, where the interleaving of these TAs expresses the same language as the global TA. Extensions of this approach are presented in Sections 4.6, 4.7 and 4.10 address potential issues and broaden the applicability of the method.

The experiments in Section 5 demonstrate the algorithm’s effectiveness and potential issues regarding scalability were identified. Active learning was suggested in order to allow for the learning of more complex tasks. The majority of the practical drawbacks are a result of the underlying Baum-Welch Algorithm that is used to learn the TAs. An advantage of my method is that the idea to restrict each agent to its local alphabet can be used to learn TAs with any TA learning method. Thus if in the future more efficient

methods are proposed for learning TAs in the single-agent case then my approach can use this in order to more efficiently learn local TAs in the multi-agent case. Therefore the main contribution of this project should still be relevant as the state of the field advances.

6.1 Reflections

Implementing the learning of local product MDPs in multi-agent settings was challenging, primarily due to using an unfamiliar C++ language. Efficiency was crucial to manage the exponential increase in runtime as the examples' size grew.

Formulating and proposing new definitions and representations for the theoretical aspects of the project was an exciting challenge. I enjoyed refining my formulations, aiming for concise expressions that precisely conveyed my intentions. It was a new and rewarding experience for me.

Unfortunately, I was unable to benchmark my work against the only other method addressing the same problem. The corresponding paper was recently released online, and the authors were not yet sharing their code. Additionally, I would have liked to conduct experiments on learning the optimal policy once the TAs were learned. However, I obtained access to the relevant algorithm's code quite late in the project's production, preventing me from including formal experiments.

6.2 Future Work

A natural next step to improve this work is to look to improve the scalability of the algorithm, potential improvements include exploring more efficient HMM learning methods for learning the Product MDP and considering alternative approaches to learning each local TA while retaining the use of local alphabets.

Incorporating active learning into the exploration stage is another way to improve scalability. By introducing a bias towards successful traces during initial exploration of unknown MDPs, we can observe more successful traces, especially in systems where successful traces occur with low probability under random exploration.

Additionally, demonstrating the effectiveness of transfer learning is a possibility. This involves leveraging existing estimates of each local product MDP as initial estimates, focusing on relearning only the parts that have changed for each local TA. Transfer learning is particularly useful when only the sub-tasks of specific agents have changed, eliminating the need to relearn product MDPs for every agent and instead updating only the relevant ones.

References

- [1] Abate, A., Almula, Y., Fox, J., Hyland, D., and Wooldridge, M. “Learning Task Automata for Reinforcement Learning using Hidden Markov Models”. *arXiv preprint arXiv:2208.11838* (2022) (cit. on pp. 5–8, 13, 15, 16, 18, 37).
- [2] Anderson, B. and Moore, A. “Active Learning for Hidden Markov Models: Objective Functions and Algorithms”. *Proceedings of the 22nd International Conference on Machine Learning*. 2005, pp. 9–16 (cit. on p. 39).
- [3] Araki, B., Li, X., Vodrahalli, K., DeCastro, J., Fry, M., and Rus, D. “The logical options framework”. *International Conference on Machine Learning*. PMLR. 2021, pp. 307–317 (cit. on p. 5).
- [4] Ardon, L., Furelos-Blanco, D., and Russo, A. *Learning Reward Machines in Cooperative Multi-Agent Tasks*. 2023 (cit. on pp. 6, 33, 37, 40).
- [5] Baum, L. E. and Petrie, T. “Statistical Inference for Probabilistic Functions of Finite State Markov Chains”. *Annals of Mathematical Statistics* 37 (1966), pp. 1554–1563 (cit. on p. 14).
- [6] Camacho, A., Icarte, R., Klassen, T., Valenzano, R., and McIlraith, S. “LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning”. 2019, pp. 6065–6073 (cit. on p. 5).
- [7] Christoffersen, P. J. K., Li, A. C., Icarte, R. T., and McIlraith, S. A. *Learning Symbolic Representations for Reinforcement Learning of Non-Markovian Behavior*. 2023 (cit. on p. 5).

- [8] De Giacomo, G. and Vardi, M. Y. “Linear Temporal Logic and Linear Dynamic Logic on Finite Traces”. *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. 2013, pp. 854–860 (cit. on p. 10).
- [9] Hasanbeig, M., Abate, A., and Kroening, D. “Logically-Correct Reinforcement Learning”. *CoRR* abs/1801.08099 (2018) (cit. on p. 5).
- [10] Hopcroft, J. “AN $n \log n$ ALGORITHM FOR MINIMIZING STATES IN A FINITE AUTOMATON”. *Theory of Machines and Computations*. Ed. by Kohavi, Z. and Paz, A. Academic Press, 1971, pp. 189–196 (cit. on p. 16).
- [11] Icarte, R. T., Klassen, T., Valenzano, R., and McIlraith, S. “Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning”. *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Dy, J. and Krause, A. Vol. 80. 2018, pp. 2107–2116 (cit. on p. 5).
- [12] Icarte, R. T., Klassen, T. Q., Valenzano, R., and McIlraith, S. A. “Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning”. *Journal of Artificial Intelligence Research* 73 (2022), pp. 173–208 (cit. on p. 5).
- [13] Jothimurugan, K., Bansal, S., Bastani, O., and Alur, R. “Compositional Reinforcement Learning from Logical Specifications”. *CoRR* abs/2106.13906 (2021) (cit. on p. 5).
- [14] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. “Human-level control through deep reinforcement learning”. *Nature* 518.7540 (2015), pp. 529–533 (cit. on p. 4).
- [15] Neary, C., Verginis, C., Cubuktepe, M., and Topcu, U. “Verifiable and Compositional Reinforcement Learning Systems”. *Proceedings of the International Conference on Automated Planning and Scheduling* 32.1 (2022), pp. 615–623 (cit. on p. 5).

- [16] Neary, C., Xu, Z., Wu, B., and Topcu, U. “Reward machines for cooperative multi-agent reinforcement learning”. *arXiv preprint arXiv:2007.01962* (2020) (cit. on pp. 5, 6, 18, 19, 21, 24, 25, 29, 33, 40).
- [17] Rabin, M. O. and Scott, D. “Finite Automata and Their Decision Problems”. *IBM Journal of Research and Development* 3.2 (1959), pp. 114–125 (cit. on p. 16).
- [18] Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. A Bradford Book, 2018 (cit. on p. 4).
- [19] Thiebaux, S., Gretton, C., Slaney, J., Price, D., and Kabanza, F. “Decision-Theoretic Planning with non-Markovian Rewards”. *J. Artif. Intell. Res. (JAIR)* 25 (2006), pp. 17–74 (cit. on p. 5).
- [20] Toro Icarte, R., Waldie, E., Klassen, T., Valenzano, R., Castro, M., and McIlraith, S. “Learning Reward Machines for Partially Observable Reinforcement Learning”. *Advances in Neural Information Processing Systems*. Ed. by Wallach, H., Larochelle, H., Beygelzimer, A., d’Alché-Buc, F., Fox, E., and Garnett, R. Vol. 32. 2019 (cit. on p. 5).
- [21] Tziola, A. A. and Loizou, S. G. *Autonomous Task Planning for Heterogeneous Multi-Agent Systems*. 2022 (cit. on p. 5).
- [22] Xu, Z., Gavran, I., Ahmad, Y., Majumdar, R., Neider, D., Topcu, U., and Wu, B. “Joint Inference of Reward Machines and Policies for Reinforcement Learning”. *Proceedings of the International Conference on Automated Planning and Scheduling* 30.1 (2020), pp. 590–598 (cit. on p. 5).