

Sample Efficient Reinforcement Learning by Automatically Learning to Compose Subtasks

Shuai Han
Utrecht University
Utrecht, the Netherlands
s.han@uu.nl

Mehdi Dastani
Utrecht University
Utrecht, the Netherlands
m.m.dastani@uu.nl

Shihan Wang
Utrecht University
Utrecht, the Netherlands
s.wang2@uu.nl

ABSTRACT

Improving sample efficiency is central to Reinforcement Learning (RL), especially in environments where the rewards are sparse. Some recent approaches have proposed to specify reward functions as manually designed or learned reward structures whose integrations in the RL algorithms are claimed to significantly improve the learning efficiency. Manually designed reward structures can suffer from inaccuracy and existing automatically learning methods are often computationally intractable for complex tasks. The integration of inaccurate or partial reward structures in RL algorithms fail to learn optimal policies. In this work, we propose an RL algorithm that can automatically structure the reward function for sample efficiency, given a set of labels that signify subtasks. Given such minimal knowledge about the task, we train a high-level policy that selects optimal sub-tasks in each state together with a low-level policy that efficiently learns to complete each sub-task. We evaluate our algorithm in a variety of sparse-reward environments. The experiment results show that our approach significantly outperforms the state-of-art baselines as the difficulty of the task increases.

KEYWORDS

Reinforcement learning, Sample efficiency, Subtasks

ACM Reference Format:

Shuai Han, Mehdi Dastani, and Shihan Wang. 2024. Sample Efficient Reinforcement Learning by Automatically Learning to Compose Subtasks. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024)*, Auckland, New Zealand, May 6 – 10, 2024, IFAAMAS, 10 pages.

1 INTRODUCTION

As a powerful technique to optimize the intelligent behaviors of agents, reinforcement learning (RL) has been applied in a variety of different domains, such as traffic signal control [9, 30], chemical structure prediction [33, 50], radio resource management [13, 56] and games [38]. The successful training of RL agents often relies on handcrafted reward functions based on domain knowledge, which allows agents to receive immediate reward signals. Without those handcrafted signals, the sparse rewards can result in RL algorithms suffering from low sample efficiency [1, 22].

Specifying the structure for a reward function with high-level logical languages or finite state machines has shown to be beneficial for improving the sample efficiency of RL algorithms [6, 7, 31]. Recently, QRM [27] has been proposed for RL agents to exploit the

specified reward structure using a reward machine. For example, when the task for the agent is to bring the coffee to the office, the corresponding reward structure is that the agent will receive reward if it first picks up the coffee and then arrives at the office. By exposing this reward structure as a reward machine and utilizing it for training the agent, the sample efficiency of RL algorithms can be significantly improved [54]. It should be noted that such specified reward function can also be used for other purposes such as task decomposition [28], reward shaping [7] and exploration [4] to make the RL algorithm more efficient.

However, for complex applications, the reward structure is not always available due to a lack of sufficient domain knowledge. As emphasized in [23], high-level objectives corresponding to subtasks can often be identified by passing through designated and semantically distinguishable states of the environment. Nevertheless, how those high-level subtasks structure the reward is not straightforward to be handcrafted [51]. Oftentimes, such high-level knowledge is implicit and unknown to the learning agent [53]. In such cases, it is crucial to automatically learn the structure of reward function based on the limited domain knowledge. To solve this problem, previous work has proposed to use automata learning to infer a automaton model to describe and exploit the reward function structure [23, 53]. However, learning an exact automaton model from trace data is a NP-complete problem [20]. Although heuristic methods can be used to speed up the learning [42], inferring an automaton that is representative to the reward structure relies on trace data with adequate exploration. When the exploration of agent is inadequate, the automaton model derived from the incomplete trace data could be either inaccurate or partial, which leads to the RL algorithm learning sub-optimal policies or even failing to learn.

In this paper, we propose a novel RL algorithm, which we call Automatically Learning to Compose Subtasks (ALCS), to automatically exploit a minimal domain knowledge to structure the reward function for sample efficiency. Given such minimal knowledge to signify subtasks, the basic idea of ALCS is to learn the best sequences of subtasks with respect to the environment reward. To realize this, we develop a framework with two-level hierarchy of policy learning. The low-level policy learns to take the next action toward completing the given subtask, while the high-level policy learns to specify a subtask to be achieved at each time step to the low-level policy. There are two main characteristics in the high-level policy learning. One is that the next subtask is selected based on the sequence of completed subtasks, which allows our method to be more accurate to make decisions to compose subtasks. Another characteristic is that during training the high-level subtask selection is modified based on the actually achieved subtasks, which can ensure that the high-level policy does not miss any necessary subtasks in the

process of reinforcing subtask selection based on environmental rewards.

This paper makes the following contributions: 1) we design a two-level RL learning framework to accurately and automatically discover and exploit reward structures through composition of subtasks, which improves sample efficiency in sparse-reward environments, 2) we design high-level subtask modification mechanism to further improve the overall policy learning, and 3) we verify the performance of our approach on 8 sparse-reward environments and show the interpretability of the agent by producing the composed sequences of subtasks. When the difficulty of tasks increases, our method produces a significant improvement over the previous most sample-efficient methods given the same minimal domain knowledge on such domain.

2 RELATED WORK

Improving sample efficiency is a key challenging problem for RL algorithms. Among the plethora of work on sample efficiency [21, 32, 36, 39, 54], we summarize the literature in three subfields, which our approach either builds on or closely relates to.

Specifying the structure of reward function. Recently, using domain knowledge to specify a reward structure that can be integrated into the RL algorithm has achieved significant improvement on the learning efficiency in sparse-reward environments. There are two major ways to provide this specified structure: Temporal Logics [45] and Mealy Machines [37]. With a logic specification for task, Temporal Logics have been applied to synthesize policies [5, 24] and shape reward [6, 31] for sample efficiency. By specifying reward structure with Mealy Machines, named reward machines, QRM [27, 28] is proposed to improve the sample efficiency for RL by exposing reward structure to the learning agent. The integration of reward machines in reinforcement learning has lead to a series of proposals for exploration [4], reward shaping [7], offline learning [47] and multi-agent learning [12, 40]. In line with these works, we also assume the domain knowledge is helpful in specifying a reward structure for sample efficiency. Our method is different from these methods in terms of the problem setting and solution. In our setting, the required domain knowledge is limited to the specification of the subtasks and not their ordering structures such as in [7, 28, 31]. In our method, the ordering structure can be automatically learned, rather than manually designed.

Learning the reward structure. When the domain knowledge minimally structures the reward function, (i.e., only specifying the possible subtasks), there are previous methods to learn the unknown reward structure for the task from trace data. LRM [51] first proposed to automatically learn a reward machine to specify the reward structure in partially observable settings. Moreover, JIRP [53] learns a smallest reward machine with automata learning techniques [29, 41]. SRM [11] considers randomness in environment rewards and learns a stochastic reward machine for task structure. Another research route is to use an automaton model to structure the reward of a task, instead of using it as a reward machine. ISA [18] learns an automaton whose transitions are subgoals given by domain knowledge. DeepSynth [23] employs an automata synthesis method to automatically reveal the sequential structure of given high-level objectives.

Similar to the above mentioned approaches, our method also assumes the availability of limited domain knowledge for improving sample efficiency of RL. The difference is that our approach does not learn an exact automata model to be integrated into the RL algorithm for sample efficiency. The proposals to learn an exact automata model from the trace data is NP-complete [20]. Besides, when the learned automaton model is inaccurate or partial for revealing complex reward structure, the integrated RL algorithm may fail to learn the optimal policy. In contrast, our approach avoids solving an NP-complete problem by using a high-level policy, which is induced directly from the real environment rewards. This allows our high-level policy to more accurately capture the reward structure.

Goal-conditioned reinforcement learning. Goal-conditioned reinforcement learning (GCRL) trains an agent to achieve different goals [8, 10, 35] by training a goal-conditioned policy. The intrinsic rewards provided by goal accelerate learning [1]. The training of goal-conditioned policy can be improved with efficient augmentation, such as curiosity exploration [17], contrastive representations [16] and causal relations [15]. Our approach follows the general training paradigm of the GCRL summarized in [10]. From the perspective of this paradigm, we propose to learn to prioritize the goal selection based on the sequence of history goals.

Hierarchical reinforcement learning. Hierarchical Reinforcement Learning (HRL) methods exploit temporal abstraction [44] or spatial abstraction [2, 55] to improve sample efficiency of RL. When designing the two-level policy, we are inspired by HRL with temporal abstraction, such as HAM [43], MAXQ [14] and h-DQN [34]. However, those methods abstract the environment as a Semi-MDP [49], which specifies different time scales for different level of policies. Instead, our high-level and low-level policies make decisions with the same time scale, i.e., at each time step high-level and low-level policies make joint decisions to produce actions. An HRL method with a similar decision-making process to ours is Interrupting Options [49], in whose extreme case the option is interrupted at every step. Another difference is that our high-level policy takes into account the sequence of historically completed subtasks when making decisions. Such a non-Markovian high-level policy is helpful to structure rewards in a more accurate way. While HRL approaches select subtasks based on the environment state, which leads to HRL being inappropriate for solving the problem of structuring rewards using domain knowledge.

3 PROBLEM SETTING

The RL problem considers an agent interacting with an unknown environment [48]. Such environment can be modeled as a Markov Decision Process (MDP), $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma)$ where \mathcal{S} is a finite set of states, \mathcal{A} is a finite set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a transition function, $\gamma \in [0, 1]$ is a discount factor and $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is a reward function. An agent employs a deterministic policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ to interact with the environment. At a time step t , the agent takes action $a_t = \pi(s_t)$ according to the current state s_t . The environment state will transfer to next state s_{t+1} based on the transition probability \mathcal{T} . The agent will receive the reward $r_t = R(s_t, a_t, s_{t+1})$. Then, the next round of interaction begins. The

goal of this agent is to find the optimal policy π^* that maximizes the expected return: $\pi^* = \text{argmax}_\pi \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | \pi]$.

Q-learning [52] is a well-known RL algorithm that obtains the optimal policy by learning its Q function. The Q function for policy π on a state-action pair (s, a) is defined as follows.

$$Q(s, a) = \mathbb{E}_{\pi, \mathcal{T}} [\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) | s_0 = s, a_0 = a] \quad (1)$$

In Q-learning, the agent learns to estimate $Q(s, a)$ by updating it based on the experience (s_t, a_t, s_{t+1}, r_t) . After the estimated Q function converges to optimal Q^* , the corresponding policy $\pi(s) = \text{argmax}_a Q^*(s, a)$ at any state $s \in \mathcal{S}$ is an optimal policy.

In this work, we extend the standard RL problem with domain knowledge consisting of a finite vocabulary set \mathcal{P} and a labeling function $L : \mathcal{S} \rightarrow \mathcal{P} \cup \{\emptyset\}$. A vocabulary $p \in \mathcal{P}$ is seen as a subtask that is potentially helpful for obtaining rewards in the environment. Given \mathcal{P} and the labeling function L , $L(s) = p$ means that p is the subtask that is achieved at state s and $L(s) = \emptyset$ means that no subtask is achieved at state s . Note that $L(s)$ does not represent the subtasks that have been achieved prior to state s .

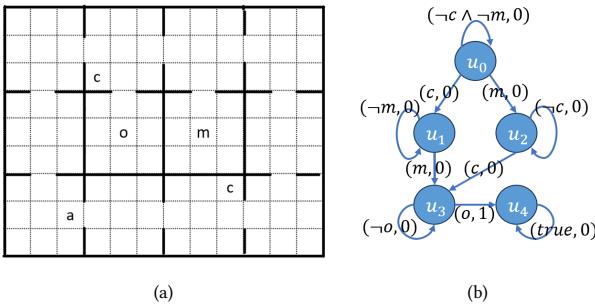


Figure 1: (a) *Coffee&mail* task on *OfficeWorld* domain, where ‘a’, ‘c’, ‘m’ and ‘o’ in the figure indicates the positions of the agent, coffee, mail and office respectively. In this task, the agent is rewarded only when it arrives at the office after taking a coffee and a mail. (b) Reward machine introduced by [28] to expose the reward structure of this task to RL agent.

A specific example is in the *OfficeWorld* domain [26, 27]. As shown in Figure 1(a), the vocabulary set $\{c, m, o\}$ specifies three possible subtasks in this environment. During interaction with the environment, agents can use L to detect whether a subtask has been completed at s . For example, $L(s) = c$ means the subtask ‘picking up coffee’ is achieved by the agent at s . Likewise, $L(s) = o$ means at state s the agent finishes the subtask ‘arriving at office’. $L(s)$ could also be \emptyset , which means no subtask achieved at s .

In recent literature, the solution to such problem is to specify [27, 28] or learn [23, 53] a reward machine, which is defined as a tuple $< U, u_0, F, \delta_u, \delta_r >$ where U is a finite set of states, u_0 is an initial state, F is a finite set of terminal states, δ_u is the state-transition function and δ_r is the state-reward function [28]. An example of a reward machine for *Coffee&mail* task is shown in 1(b). An RL agent with this reward machine starts from u_0 at the

beginning of an episode. If at any MDP state s that $L(s) = c$, then the RL agent moves to $u_1 \in U$ with reward 0. Similarly, if at $u_3 \in U$ the RL agent encounters an MDP state s that $L(s) = o$ then the RL agent moves to terminal state u_4 with reward 1. With such a reward machine, previous methods such as [23, 27, 53] learn Q -values over the cross-product $Q(s, u, a)$, which allows the agent to consider the MDP state s and RM state u to select the next action a [28].

Different with previous works [5, 27, 28], in our setting, less domain knowledge about the task is required and given. The knowledge given by \mathcal{P} and L is limited to subtasks and not to, for example, their ordering structures. With such a minimal knowledge to the task, the agent does not know the information how the subtasks contributes to rewards before training. In our setting, this information will be learned during the training.

4 METHODOLOGY

In this section we present our two-level policy design and the detailed training for the policies.

4.1 Two-level policy formalization

When the domain knowledge can specify and structure the reward function, the RL agent can use them to improve sample efficiency [28]. When the domain knowledge minimally specifies and structures the reward function, as in our case, an inertial solution is needed to learn an exact automaton model describing the reward structure and then use it to improve RL [23, 53]. However, learning an exact automaton model from trace data is known to be NP-complete [20], which could make the algorithm computationally intractable. Besides, when the learned automaton model is inaccurate or partial to specify the reward function, the RL agent may fail to learn optimal policy. Therefore, instead of learning an exact model with domain knowledge, in this work we propose a more efficient way to integrate domain knowledge into the learning procedure of an RL algorithm.

In the framework of ALCS, a high-level policy is designed to select next subtasks to be achieved based on the environment state and the sequence of subtasks that have already achieved in the history this episode. We denote it as $\pi_h : \mathcal{S} \times \mathcal{P}^* \rightarrow \mathcal{P}$, where \mathcal{P}^* is the Kleene closure on \mathcal{P} . For example, in Figure 1 where $\mathcal{P} = \{c, m, o\}$, the Kleene closure on \mathcal{P} is $\mathcal{P}^* = \{\emptyset, c, m, o, cc, cm, co, mc, mm, mo, oc, om, oo, ccc,\}$. Given a state s , π_h selects the next subtask p by:

$$p = \pi_h(s, p^*) \quad (2)$$

where the sequence $p^* \in \mathcal{P}^*$ represents the order of subtasks have been temporally achieved in the history of a given episode. We believe that the completed subtasks and their order are key information for π_h to select the next subtask to be achieved. Therefore, the agent will benefit from training the policy π_h with such sequence being considered.

On the other hand, a low-level policy, denoted as $\pi_l : \mathcal{S} \times \mathcal{P} \rightarrow \mathcal{A}$, is designed to learn to achieve the selected subtask efficiently. Taking the current state and a given subtask, π_l decides an environment action to achieve the subtask. π_l selects actions by:

$$a = \pi_l(s, p) \quad (3)$$

With the two-level policies, our agent interacts with the environment as follows. At the beginning of an episode, the agent initializes an empty sequence p^* to store the achieved subtasks in the environment. At each time step t , the agent first employs the high-level policy to select a subtask p_t to be achieved based on the current state s_t and the achieved subtask sequence p_t^* , i.e., $p_t = \pi_h(s_t, p_t^*)$. Then, the agent uses the low-level policy to choose action $a_t = \pi_l(s_t, p_t)$ to interact with the environment and receive the reward r_t the next state s_{t+1} . Moreover, if a subtask is achieved in s_{t+1} , i.e., $L(s_{t+1}) \neq \emptyset$, then the achieved subtask will be appended into the sequence:

$$p_{t+1}^* = \begin{cases} p_t^* \oplus L(s_{t+1}) & \text{if } L(s_{t+1}) \neq \emptyset \\ p_t^* & \text{otherwise} \end{cases} \quad (4)$$

where \oplus represents appending $L(s_{t+1})$ into the end of sequence p_t^* .

Following Q-learning, our π_h and π_l can select subtasks and actions by doing argmax on their corresponding Q functions. We next describe the definition of their Q functions and the detailed corresponding training process.

4.2 Low-level training

In order to define the low-level Q function, we first define the low-level reward function. The goal of low-level policy π_l is to achieve the given subtask $p \in \mathcal{P}$. Therefore, in our design, the low-level policy does not directly maximize the expected return from the environment reward function R . For a certain $p \in \mathcal{P}$, $\pi_l(s, p)$ is trained with the following rewards:

$$R^p(s_t, a_t, s_{t+1}) = \begin{cases} 1 & \text{if } p = L(s_{t+1}) \text{ and } p \neq L(s_t) \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

According to Equation (5), it can be known that for different subtasks, π_l is trained with different reward functions. Given Equation (5), we define the low-level Q function $Q_l(s, p, a)$ as the expected return based on the subtask p :

$$Q_l(s, p, a) = \mathbb{E}_{\pi_l, \mathcal{T}} \left[\sum_{t=0}^{\infty} \gamma^t R^p(s_t, a_t, s_{t+1}) | s_0 = s, a_0 = a \right] \quad (6)$$

With the defined low-level Q_l in Equation (6), π_l can select actions by doing argmax operation: $a_t = \text{argmax}_a Q_l(s_t, p, a)$ at time step t . Then we can train π_l by updating Q_l with experience $(s_t, a_t, s_{t+1}, r_t^p)$ using Q-learning, where r_t^p is the reward value computed from Equation (5) with the current subtask p to be achieved.

We observe that the reward function in Equation (5) is computed by labeling function L . Inspired by counterfactual experiences [28], to improve the training efficiency, we use L to generate multiple experience for different subtasks to update Q_l , instead of using a single experience $(s_t, a_t, s_{t+1}, r_t^p)$ to update it. To do so, for a single transition (s_t, a_t, s_{t+1}) sampled from the environment, we can generate a set of experiences: $\{(s_t, a_t, s_{t+1}, r_t^p)\}$ for all $p \in \mathcal{P}$, where $r_t^p = R^p(s_t, a_t, s_{t+1})$ is the reward value with respect to subtask p at time step t .

Algorithm 1 describes the detailed practice for updating Q_l with multiple experiences generated from single transition. Given an input transition (s_t, a_t, s_{t+1}) , Algorithm 1 in Lines 2 ~ 7 assigns different rewards following Equation 5 for all possible $p \in \mathcal{P}$ to this transition, which yields a collection of experiences. Then those experiences will be used to update Q_l in Lines 10 ~ 17.

Algorithm 1 Generating multiple experiences and updating Q_l .

Input: Transition (s_t, a_t, s_{t+1}) , original Q_l , learning rate α
Output: Updated Q_l

```

1: Initialize experiences  $\leftarrow \{\}$ 
2: for  $p \in \mathcal{P}$  do
3:   if  $p = L(s_{t+1})$  and  $p \neq L(s_t)$  then
4:      $r_t^p \leftarrow 1$ , done  $\leftarrow \text{True}$ 
5:   else
6:      $r_t^p \leftarrow 0$ , done  $\leftarrow \text{False}$ 
7:   end if
8:   Add tuple  $(s_t, a_t, s_{t+1}, r_t^p, p, \text{done})$  into experiences
9: end for
10: for  $(s, a, s', r, p, \text{done})$  in experiences do
11:   if done then
12:      $y \leftarrow r$ 
13:   else
14:      $y \leftarrow r + \gamma \max_{a'} Q_l(s', p, a')$ 
15:   end if
16:    $Q_l(s, p, a) \leftarrow (1 - \alpha)Q_l(s, p, a) + \alpha \cdot y$ 
17: end for

```

After sufficiently updating for Q_l , given a subtask p , π_l can repeatedly perform this subtask. But this is generally not enough to maximize the environment rewards. So we also need to train the π_h , which is responsible for learning to compose subtasks in any order to maximize the environment rewards.

4.3 High-level training

The goal of the high-level policy π_h is to compose subtasks for the original task in MDP. So we use the environment original reward to train the π_h . The corresponding Q function for π_h is denoted as $Q_h(s, p^*, p)$, which is defined as the expected return from the environment following π_h and π_l if the agent selects p as a subtask given s and p^* :

$$Q_h(s, p^*, p) = \mathbb{E}_{\pi_l, \pi_h, \mathcal{T}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) | s_0 = s, a_0 = \pi_l(s, p) \right] \quad (7)$$

With the high-level function defined in Equation (7), π_h selects a subgoal p_t by doing the argmax operation: $p_t = \text{argmax}_p Q_h(s_t, p_t^*, p)$ for given s_t and p_t^* .

In the general RL methods, Q function should be updated based on the outputs from the corresponding policy. However, when this approach is applied to our high-level training, it may introduce the problem of multiple subtasks being completed under one target subtask. This can lead to decisions about key subtasks not being learned. For example, a sub-goal p_t is chosen by $\pi_h(s_t, p_t^*)$. When $\pi_l(s_t, p_t)$ takes actions to achieve p_t , another subtask p' could be completed. In this case, if p' is the key subtask that brings reward in future time step, the importance of p' to reward cannot be learned because $\pi_h(s_t, p_t^*)$ never choose p' as its decision.

To solve this problem, we propose to generate the high-level experiences based on the actual completed subtask in the environment (subtask p' in the above example), instead of based on the subtask selected by the high-level policy (sub-task p in the example). In other words, the actual completed task will be rewarded and used in the corresponding data $((s_t, p_t^*), p', (s_{t+1}, p_{t+1}^*), r_t)$ for the update. The basic idea is that whenever a subtask is detected to have been achieved by the labeling function, we assume that π_h has chosen this subtask. And experiences will be generated to

$L(s_9) = c, L(s_{15}) = m, L(s_{27}) = o$	Generate experience
$((s_0, \emptyset), c, (s_1, \emptyset), r_0)$	Assumption: c
$((s_1, \emptyset), c, (s_2, \emptyset), r_1)$	
\dots	
$((s_8, \emptyset), c, (s_9, c), r_8)$	Assumption: c
$((s_9, c), m, (s_{10}, c), r_9)$	Assumption: m
$((s_{10}, c), m, (s_{11}, c), r_{10})$	
\dots	
$((s_{14}, c), m, (s_{15}, cm), r_{14})$	Assumption: m
$((s_{15}, cm), o, (s_{16}, cm), r_{15})$	Assumption: o
$((s_{16}, cm), o, (s_{17}, cm), r_{16})$	
\dots	
$((s_{26}, cm), o, (s_{27}, cmo), r_{27})$	

Figure 2: Following the environment and task of Figure 1, here is an example about how we generate the high-level experiences in an episode for updating Q_h . As shown in the table, each row contains the corresponding experience data at that time step. The experience is denoted as: $((s_t, p_t^*), p_t, (s_{t+1}, p_{t+1}^*), r_t)$, where p_t here is the assumed selected subtask and r_t is reward from the environment. In this episode, subtask c , m , and o are achieved at time step 9, 15, and 27, respectively. Then, the assumed subtask p_t selected by π_h are c from time step $0 \sim 8$, m from time step $9 \sim 14$, and o from time step $15 \sim 27$.

update Q_h based on this assumption. An example for generating experiences in an episode is shown in Figure 2.

We also would like to clarify that in our design the exploration of high-level policy is stochastic. In fact, in our two-level policy design, as long as action selection in low-level is stochastic, high-level policy will also perform stochastic exploration. Because the stochastic actions cause the subtasks being achieved stochastically, the assumed high-level choice will be stochastic as well.

4.4 Overall algorithm

Algorithm 2 shows the overall training procedure of our method. When an episode begins, we initialize some variables in Lines 2 ~ 4, where $experience_h$ is the experiences for this episode to update Q_h , e_temp is a temporary experience set, and p_t^* stores the sequence of achieved subtasks at the time step t . After the initialization for an episode, the algorithm will repeat the process in Line 6 ~ 36 until the episode terminates. In each episode, the agent first choose a subtask p_t in Line 6. Then an action a_t is selected following the ϵ -greedy exploration in Line 7 ~ 11, where $rand()$ represents a function for sampling a variable between 0 and 1 uniformly. After executing a_t , the Q_l is updated using Algorithm 1 with respect to the current transition in Line 13. Then, high-level experiences are generated in Line 14 ~ 28. The experiences e in line 15 and Line 21 are differentiated according to whether there is a subtask being achieved at current time step. If a subtask is achieved (i.e., $L(s_{t+1}) \neq \emptyset$), this achieved subtask, denoted as p_{act} in Line 19, will be appended into p_t^* in Line 20. Besides, p_{act} will be treated as

Algorithm 2 Automatically Learning to Compose Subtasks.

```

Input: Total episode  $M$ , learning rate  $\beta$ , exploration rate  $\epsilon$ 
Output:  $Q_l, Q_h$ 
1: for episode = 1 →  $M$  do
2:   Reset Env and get  $s_0, t \leftarrow 0$ 
3:    $experience\_h \leftarrow \{\}$ ,  $e\_temp \leftarrow \{\}$ 
4:    $p_t^* \leftarrow []$ 
5:   while not terminal do
6:     Obtain subtask  $p_t = argmax_p Q_h(s_t, p_t^*, p)$ 
7:     if  $rand() > \epsilon$  then
8:       Take the action  $a_t = argmax_a Q_l(s_t, p_t, a)$ 
9:     else
10:      Take a random action as  $a_t$ 
11:    end if
12:    Execute  $a_t$  and get  $s_{t+1}$  and  $r_t$  from Env
13:    Update  $Q_l$  using Algorithm 1 with  $(s_t, a_t, s_{t+1})$ 
14:    if  $L(s_{t+1}) = \emptyset$  then
15:       $e \leftarrow ((s_t, p_t^*), p_t, (s_{t+1}, p_{t+1}^*), r_t)$ 
16:      Add experience tuple  $e$  into  $e\_temp$ 
17:       $p_{t+1}^* \leftarrow p_t^*$ 
18:    else
19:       $p_{act} \leftarrow L(s_{t+1})$ 
20:       $p_{t+1}^* \leftarrow p_t^* \oplus p_{act}$ 
21:       $e \leftarrow ((s_t, p_t^*), p_t, (s_{t+1}, p_{t+1}^*), r_t)$ 
22:      Add experience tuple  $e$  into  $e\_temp$ 
23:      for  $((s, p^*), p, (s', p^{*'}), r)$  in  $e\_temp$  do
24:         $e\_assumed \leftarrow ((s, p^*), p_{act}, (s', p^{*'}), r)$ 
25:        Add  $e\_assumed$  into  $experience\_h$ 
26:      end for
27:       $e\_temp \leftarrow \{\}$ 
28:    end if
29:    if terminal then
30:      for  $((s, p^*), p, (s', p^{*'}), r)$  in  $experience\_h$  do
31:         $y \leftarrow r + \gamma \max_{p'} Q_h(s', p^{*'}, p')$ 
32:         $Q_h(s, p^*, p) \leftarrow (1 - \beta)Q_h(s, p^*, p) + \beta \cdot y$ 
33:      end for
34:       $experience\_h \leftarrow \{\}$ 
35:    end if
36:     $t \leftarrow t + 1$ 
37:  end while
38: end for

```

the assumed subtask selected by π_h . Based on p_{act} the high-level experiences for updating are generated in Line 23 ~ 26. Finally, Q_h will be updated for each experience in Line 30 ~ 33.

5 INTERPRETABILITY

How to make the RL algorithms interpretable has widely raised concerns recently for ethical and legal considerations in application [19, 25, 46]. In this section, based on the given minimal domain knowledge, we interpret the behavior of agents trained by ALCS.

We can use a tree structure to record all the sequence of subtasks that have been achieved in the training process of ALCS, which can be used to interpret the behavior of an agent. This tree begins with a root node ‘0’. Each descendant node of the root is a subtask $p \in \mathcal{P}$. The nodes in the path from the root to a descendant can form a sequence $p^* \in \mathcal{P}^*$. In the beginning of an episode, the agent always starts from the root node ‘0’, which means no subtask has been achieved in the history of this episode. Then, if a subtask p is achieved, a new child node with respect to p is added in the tree. Besides, the reward from environment when subtask p is achieved will be recorded on the edge to the corresponding child node. Given an achieved sequence of subtask p_t^* at time step t , a node associated with the current state can be uniquely identified in the tree. We call this node as a current node at t , denoted as $N(p_t^*)$. At the start

of the episode, the agent’s current node $N(p_t^*)$ is reset to the root node, i.e. p_t^* is reset to \emptyset . An example for this is shown in Figure 3.

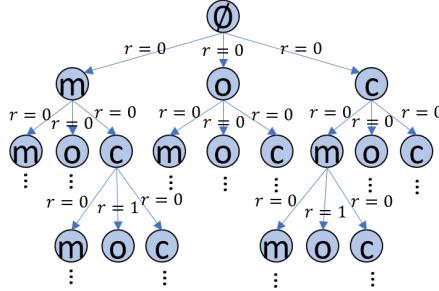


Figure 3: An example tree to record sequences of subtasks in the environment of Figure 1. This tree is generated during the training process. Except for the root, nodes in this tree refer to the subtasks specified in this environment. The edges present the transferring from one subtask to another subtask (when the parent subtask has been achieved). The reward values on the edges are sampled from the environment based on whether the main task is achieved. Following the task description in Figure 1, ‘ $r = 1$ ’ means the task is finished, i.e., the agent arrives at the office after taking both a coffee and a mail. For infinite MDP where the episode step can go to infinity, the tree could be infinite. For finite MDP where the episode step is limited, the tree is finite because the depth of the tree will be limited by the episode steps and the width will be limited by $|\mathcal{P}|$.

With this tree, the RL agent can interpret its behavior based on the given domain knowledge during execution. The interpretation consists of the following three parts.

What has already happened. At a certain time step during execution, according to the current p_t^* , we can identify the current node of the trained RL agent. Because p_t^* includes the historically achieved subtasks, which can be seen as an abstraction for the episode history based on domain knowledge. Therefore, p_t^* can be used as an interpretation of what has happened.

Current best subtask. Based on the current selected p_t , the agent can interpret its current action a_t in terms of which subtask this action is trying to complete. For example, if c is given from π_h in the example task of Figure 1, then the action from π_l can be interpreted as an action trying to complete c .

Planning for the future. After obtaining p_t , the agent can find the child node of the current node. By performing a breadth-first search (BFS) on the subtree with this child node as root, the agent can find a descendant node that can obtain the reward based on the previous record. By providing the nodes on this path, the agent can interpret an estimated future planning for the current selected subtask. Taking Figure 3 as an example, suppose current $p_t^* = \emptyset$ and $p_t = c$, then the estimated future plan is m and o , considering that the BFS algorithm will find the satisfied descendant node with minimum depth first and return it. If the BFS algorithm cannot find a satisfied descendant node in limited depth, then we say there is no interpretation to future plan for the current decision.

The advantage of this is that the interpretation takes into account both the high-level information constructed by domain knowledge (i.e., the tree) and the environment state information through $p_t = \pi(s_t, p_t^*)$. When the environment state changes, both the behavior and its interpretation could change, which makes agent more transparent on its decision making procedure. We will further demonstrate this benefit in Subsection 6.4.

6 EXPERIMENTS

This section presents the experiment settings and results¹.

6.1 Experiment settings

We evaluate our method in 8 different sparse-reward environments from the two commonly used domains, *OfficeWord* and *MineCraft* [27, 28, 53]. We introduce the characteristics of these environments and the reasons for choosing them as follows.

Coffee. Go to office after taking a coffee. The agent receives 1 reward after achieving this task.

Coffee&Mail. Go to office after taking both a coffee and a mail. The agent receives 1 reward after achieving this task. The order of taking the coffee and mail does not matter.

Collecting. Go to office after collecting four packages A, B, C and D in the corners. The agent receives 1 reward after achieving this task. The order of taking those packages does not matter.

Bonus. Same task with the Collecting environment. When agent arrive at office, it will receive 1 reward for each package, and 5 bonus reward if all packages are collected.

Plant. Get wood, use toolshed. The agent receives 1 reward after achieving this task.

Bridge. Get iron, get wood, use factory. The agent receives 1 reward after achieving this task. The iron and wood can be gotten in any order.

Bed. Get wood, use toolshed, get grass, use workbench. The agent receives 1 reward after achieving this task. The grass can be gotten at any time before using the workbench.

Gem. Get wood, use workbench, get iron, use toolshed, use axe. The agent receives 1 reward after achieving this task. The iron can be gotten at any time before using the toolshed.

In the above environments, the former four environments are from the *OfficeWord* domain, while the latter four are from *MineCraft*. These environments have increasing task complexity in their respective domains, which is helpful to show how the performance of methods changes as the task difficulty increases. Moreover, *MineCraft* domain has a larger state space than *OfficeWord*. Comparing the performance of methods in the two domains can help to demonstrate the impact of the increased state space on the performance of different methods.

In addition, Bonus is a special environment. In this environment, the agent can get 1 reward for each package, but prefers to collect all the packages. Note that the goal in the Bonus environment is too sparse for providing rewards. Thus, in this environment, there is also environment reward for completing individual subtasks (collecting one or more packages), which helps the baseline methods learn the skill of collecting packages. But at the same time those reward can lead to the agent misalignment policy from learning

¹The code is released (anonymously) at: <https://github.com/anonymousanonymouse/ALCS>.

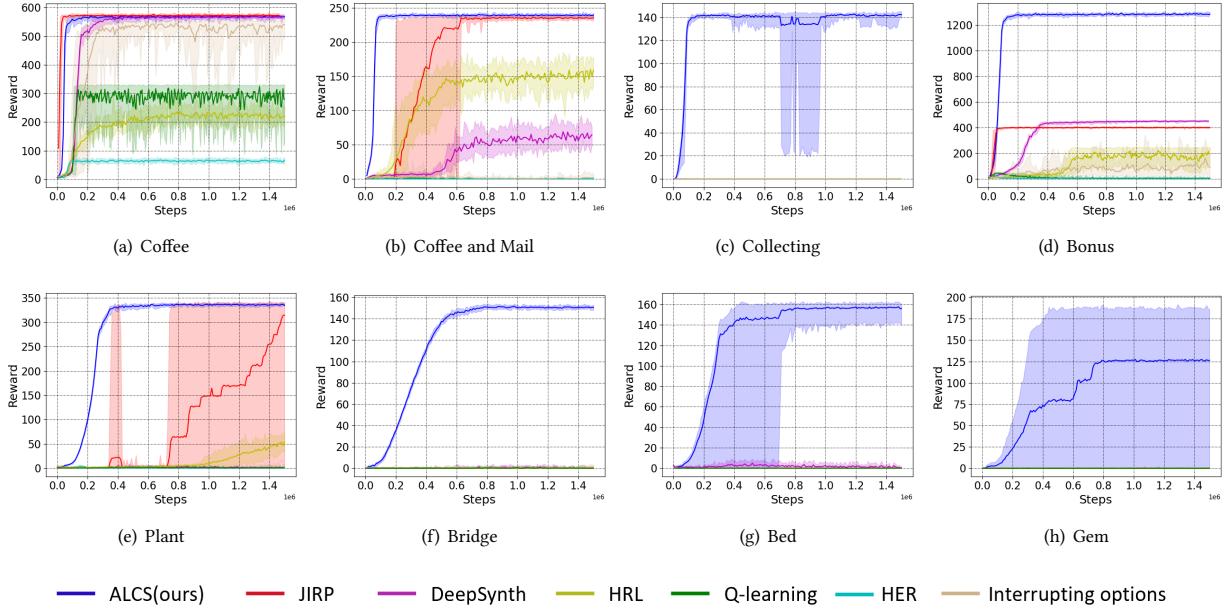


Figure 4: Learning curves of various RL algorithms on 8 environments from *OfficeWord* and *MineCraft* domains.

to collect all packages to learning to collect individual packages. Therefore in this environment, it is challenging for the RL agent to utilize the subtask reward without such policy misalignment.

We compare the following five different methods.

ALCS (ours). We implement a prototype of ALCS, which uses the Q-learning algorithm [52] to learn both the high-level and low-level policies. The learning rates for both Q_l and Q_h are 0.1. In the training, the rate for ϵ -greedy exploration is 20% to provide a stochastic exploration strategy.

JIRP. JIRP [53] is a method that infers a reward machine using the *libalf* library [3] as the RL agent interacts with the environment. We chose JIRP as a baseline because it has a similar problem setting to our work, i.e., how to automatically learn to structure the unknown reward function through a minimal domain knowledge.

DeepSynth. DeepSynth [23] is an algorithm that infers a automaton model to guide RL agent to learn policy. This method uses automata synthesis [29] as an approach to learn a high-level model. Likewise, DeepSynth has a similar problem setting to our work.

HRL. HRL [34] learns hierarchical policies in the environment. Following previous implementation [53], the option for high-level policy in HRL is defined as subtask $p \in \mathcal{P}$. The termination condition of an option is whenever the corresponding p is achieved.

Interrupting options. Interrupting options [49] is a hierarchical RL method in which the high-level policy can interrupt low-level policies and change the current subtask to be completed. We chose HRL and Interrupting options as our baselines because their policies have similar structures to our method.

HER. HER [1] is a well-known goal-conditioned RL method that trains policies to achieve different goals. The goal in HER is defined as the subtask $p \in \mathcal{P}$. Each goal-conditioned policy is trained with the same reward function as Equation (5). We choose HER as one of

our baselines because the training of HER goal-conditioned policies is similar with our low-level policy training.

Q learning. Q-learning [52] is a well-known general RL algorithm. We use the implementation version from [28].

Notably, JIRP and DeepSynth are two state-of-the-art methods in *OfficeWord* and *MineCraft* domains. Generally speaking, JIRP learns better when there is only binary reward signal in the end of episode, and DeepSynth is more adaptable in non-binary cases. Except the Q-Learning which does not require subtask, the other five baselines use equivalent domain knowledge with the proposed ALCS method. They all have access to a given set of subtasks, as well as a label function that can return to agent the identity of the subtask based on the environmental state when the subtask is completed for the first time in an episode.

6.2 Comparison

We first compare our method with baselines on 8 environments from *OfficeWord* and *MineCraft* domains to validate the superiority of ALCS. The results are shown in Figure 4. The error bounds (i.e., shadow shapes) indicate the upper and lower bounds of the performance with 20 runs excluding 2 best and 2 worst results.

The results show that our method significantly outperforms the baseline methods in all environments except Coffee where the reward structure is simple and easy to explore. In such an environment, learning an exact model with JIRP in Coffee environments to specify the reward structure is a better solution. However, when the reward structure of the task becomes complex, the sample efficiency of our method can significantly outperform methods that learn models to specify the reward structure.

Another conclusion can be drawn by comparing those methods between *OfficeWord* and *MineCraft* domains. It is well known that

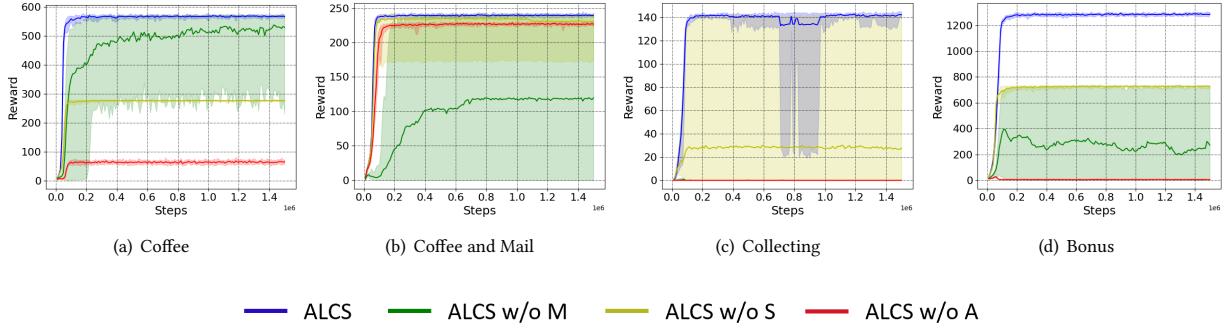


Figure 5: Learning curves for ablation experiment of ALCS on 4 environments from *OfficeWord*.

the sample efficiency of the RL algorithm can decrease on environments with large state space. Comparing the performance of those RL algorithms in *OfficeWord* domain, their performance in the *MineCraft* are reduced because of the larger state space. In contrast, our approach is less affected by increased state space compared to the baselines approaches. Nevertheless, by comparing the performance of ALCS across Bridge, Bed and Gem environments, it can be seen that the performance variance of ALCS grows with the increasing task complexity. This suggests that there is room to improve the performance of ALCS in complex tasks with large state spaces.

Furthermore, comparing with Collecting environment where all baselines totally fails on policy learning, these baselines can learn some policies for subtasks when rewards is shaped for completing the collection of one or more packages. However, these shaped rewards can misalign the learned policy from collecting all packages to collecting individual packages for baselines. In contrast, our two-layer policy learning can overcome such policy misalignment.

6.3 Ablation study

In order to explore the contribution that each proposed technique to the whole algorithm, we compare the ALCS with its ablative variants on the *OfficeWord* domain. The results are shown in Figure 5, where ‘ALCS w/o M’ stands for ALCS algorithm without multiple experiences generating for updating Q_l (as proposed in Subsection 4.2), ‘ALCS w/o S’ represents ALCS algorithm without sequence of completed subtask for π_h (as presented in Subsection 4.1), and ‘ALCS w/o A’ indicates ALCS algorithm without the assumed choice of subtask for training π_h (as presented in Subsection 4.3).

As shown in Figure 5, in case of ALCS w/o M, when not generating multiple experiences for Q_l , the performance of the algorithm drops in all environments due to insufficient learning of the low-level policy, and in Collecting environment, ALCS w/o M cannot even learn. In ALCS w/o S, when π_h learns without a sequence of completed subtask, it cannot select the optimal subtask to be achieved due to the inability to obtain sufficient information on the completed subtasks. In case of ALCS w/o A, we see that the assumed choice of the subtask is the key to high-level π_h to efficiently learn to compose subtasks. In addition, in the coffee environment, the subtask on picking coffee has a high probability of being completed while executing subtask on arriving at office. In this case,

the selection of subtask for picking a coffee will not be learned by the π_h . Therefore, even if the tasks on coffee environment are not complex, without assumed choice of subtask, it is difficult for ALCS to learn good policies.

6.4 Interpretability

In this subsection, we demonstrate the interpretability of our learned policy in the Coffee and Mail environment.

In this task, there are two possible sequences of subtasks to bring reward: $c \rightarrow m \rightarrow o$ and $m \rightarrow c \rightarrow o$. However, the optimal sequence of subtasks is different when the agent starts from different positions. As shown in Figure 6, starting from position ①, the agent takes fewer steps with the sequence $c \rightarrow m \rightarrow o$. At position ② $m \rightarrow c \rightarrow o$ is a better sequence. The previous approaches, by using an automaton model to specify the reward function, allows two possible sequences to be represented in a single model [28, 53]. Nevertheless, their models cannot explain that following which sequence of subtasks the agent will take fewer steps. For example, in the left of Figure 6, when starting with different positions ① and ②, the automaton model will fail to explain the agent will follow which sequence of subtasks for finishing the task with fewer steps.

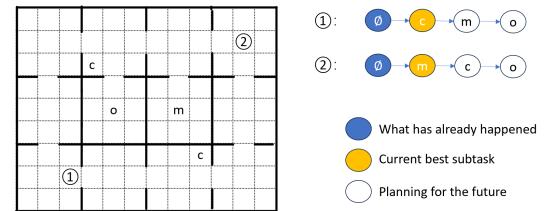


Figure 6: Interpretation experiment on Coffee and Mail environment. The three colors on the right of the figure represent the three corresponding parts of interpretation introduced in Section 5.

The high-level policy that we are learning about can address this issue to some extent, because our high-level policy can output an exact subtask to be achieved based on the environment state. As shown in Figure 6, when the agent starts in different positions, with different state being input into π_h , π_h will select different ‘current best subtask’ to achieve based on Equation (2). As shown in right

of Figure 6, ‘c’ is selected at ① and ‘m’ is selected at ②. Then, according to the recording tree in Section 5, it provides a future planning after achieving the selected subtask. These constitute a complete interpretation on how to compose subtasks exactly to finish the task with fewer steps.

7 CONCLUSION AND FUTURE WORK

In this paper, we propose ALCS, an RL algorithm to automatically structure the reward function to improve the sample efficiency in sparse-reward tasks. ALCS employs a two-level policy to learn composing subtasks and to achieve them in best order for maximizing the expected return from environments. Besides, three optimization methods are designed for this two-level policy learning framework, which includes providing information of completed subtask sequence for better high-level decision, generating multiple experience for sample efficiency, and training high-level policy with assumed selection based on the actually achieved subtasks. We show that in a variety of sparse-reward environments, ALCS significantly outperforms the state-of-the-art methods in the environments with high task difficulty. In future work, we want to explore the convergence of the proposed algorithm and to continue our efforts to automatically learn to structure the reward function. In this paper, the subtasks is given by domain knowledge that specify subtasks. In future work, we will extend our method with partial information about the specified subtasks.

ACKNOWLEDGMENTS

If you wish to include any acknowledgments in your paper (e.g., to people or funding agencies), please do so using the ‘acks’ environment. Note that the text of your acknowledgments will be omitted if you compile your document with the ‘anonymous’ option.

REFERENCES

- [1] Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight Experience Replay. 5048–5058.
- [2] Nir Baram, Tom Zahavy, and Shie Mannor. 2016. Spatio-temporal abstractions in reinforcement learning through neural encoding. (2016).
- [3] Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker, Daniel Nieder, and David R Piegdon. 2010. libalf: The automata learning framework. In *Proceedings of the 22nd International Conference on Computer Aided Verification*. 360–364.
- [4] Hippolyte Bourel, Anders Jonsson, Odalric-Ambrym Maillard, and Mohammad Sadegh Talebi. 2023. Exploration in Reward Machines with Low Regret. In *International Conference on Artificial Intelligence and Statistics*, Vol. 206. 4114–4146.
- [5] Alper Kamil Bozkurt, Yu Wang, Michael M. Zavlanos, and Miroslav Pajic. 2020. Control Synthesis from Linear Temporal Logic Specifications using Model-Free Reinforcement Learning. In *2020 IEEE International Conference on Robotics and Automation*. 10349–10355.
- [6] Ronen I. Brafman, Giuseppe De Giacomo, and Fabio Patrizi. 2018. LTLF/LDLF Non-Markovian Rewards. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. 1771–1778.
- [7] Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. 2019. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. 6065–6073.
- [8] Elliot Chané-Sane, Cordelia Schmid, and Ivan Laptev. 2021. Goal-conditioned reinforcement learning with imagined subgoals. In *International Conference on Machine Learning*. PMLR, 1430–1440.
- [9] Chacha Chen, Hua Wei, Nan Xu, Guanjie Zheng, Ming Yang, Yuanhao Xiong, Kai Xu, and Zhenhui Li. 2020. Toward A Thousand Lights: Decentralized Deep Reinforcement Learning for Large-Scale Traffic Signal Control. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*. 3414–3421.
- [10] Cédric Colas, Tristan Karch, Olivier Sigaud, and Pierre-Yves Oudeyer. 2022. Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: a short survey. *Journal of Artificial Intelligence Research* 74 (2022), 1159–1199.
- [11] Jan Corazza, Ivan Gavran, and Daniel Neider. 2022. Reinforcement learning with stochastic reward machines. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 6429–6436.
- [12] Michael Dann, Yuan Yao, Natasha Alechina, Brian Logan, and John Thangarajah. 2022. Multi-Agent Intention Progression with Reward Machines. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*. 215–222.
- [13] James Delaney, Steve Dowey, and Chi-Tsun Cheng. 2023. Reinforcement-Learning-Based Robust Resource Management for Multi-Radio Systems. *Sensors* 23, 10 (2023), 4821.
- [14] Thomas G Dietterich. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of artificial intelligence research* 13 (2000), 227–303.
- [15] Wenhao Ding, Haohong Lin, Bo Li, and Ding Zhao. 2022. Generalizing goal-conditioned reinforcement learning with variational causal reasoning. *Advances in Neural Information Processing Systems* 35 (2022), 26532–26548.
- [16] Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Russ R Salakhutdinov. 2022. Contrastive learning as goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems* 35 (2022), 35603–35620.
- [17] Meng Fang, Tianyi Zhou, Yali Du, Lei Han, and Zhengyou Zhang. 2019. Curriculum-guided hindsight experience replay. *Advances in neural information processing systems* 32 (2019).
- [18] Daniel Furelos-Blanco, Mark Law, Alessandra Russo, Krysia Broda, and Anders Jonsson. 2020. Induction of subgoal automata for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3890–3897.
- [19] Claire Glanois, Paul Weng, Matthieu Zimmer, Dong Li, Tianpei Yang, Jianye Hao, and Wulong Liu. 2021. A survey on interpretable reinforcement learning. *arXiv preprint arXiv:2112.13112* (2021).
- [20] E. Mark Gold. 1978. Complexity of Automaton Identification from Given Data. *Inf. Control.* 37, 3 (1978), 302–320.
- [21] Zijing Guo, Chendie Yao, Yanghe Feng, and Yue Xu. 2022. Survey of Reinforcement Learning based on Human Prior Knowledge. *Journal of Uncertain Systems* 15, 01 (2022), 2230001.
- [22] Abhishek Gupta, Aldo Pacchiano, Yuexiang Zhai, Sham Kakade, and Sergey Levine. 2022. Unpacking reward shaping: Understanding the benefits of reward engineering on sample complexity. *Advances in Neural Information Processing Systems* 35 (2022), 15281–15295.
- [23] Mohammadhosein Hasanbeig, Natasha Yogananda Jeppu, Alessandro Abate, Tom Melham, and Daniel Kroening. 2021. DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence*. 7647–7656.
- [24] Mohammadhosein Hasanbeig, Yiannis Kantaros, Alessandro Abate, Daniel Kroening, George J. Pappas, and Insup Lee. 2019. Reinforcement Learning for Temporal Logic Control Synthesis with Probabilistic Satisfaction Guarantees. In *58th IEEE Conference on Decision and Control*. 5338–5343.
- [25] Alexandre Heuillet, Fabien Couthouis, and Natalia Díaz-Rodríguez. 2021. Explainability in deep reinforcement learning. *Knowledge-Based Systems* 214 (2021), 106685.
- [26] Rodrigo Toro Icarte. 2022. *Reward Machines*. Ph.D. Dissertation. University of Toronto, Canada.
- [27] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. 2018. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80. 2112–2121.
- [28] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. 2022. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. *J. Artif. Intell. Res.* 73 (2022), 173–208.
- [29] Natasha Yogananda Jeppu, Thomas Melham, Daniel Kroening, and John O’Leary. 2020. Learning concise models from long execution traces. In *57th ACM/IEEE Design Automation Conference (DAC)*. 1–6.
- [30] Qize Jiang, Minhao Qin, Shengmin Shi, Weiwei Sun, and Baihua Zheng. 2022. Multi-Agent Reinforcement Learning for Traffic Signal Control through Universal Communication Method. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*. 3854–3860.
- [31] Yuqian Jiang, Suda Bharadwaj, Bo Wu, Rishi Shah, Ufuk Topcu, and Peter Stone. 2021. Temporal-Logic-Based Reward Shaping for Continuing Reinforcement Learning Tasks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence*. 7995–8003.
- [32] Mu Jin, Zhihao Ma, Kebing Jin, Hankz Hankui Zhuo, Chen Chen, and Chao Yu. 2022. Creativity of ai: Automatic symbolic option discovery for facilitating deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 7042–7050.
- [33] Hwanhee Kim, Soohyun Ko, Byung Ju Kim, Sung Jin Ryu, and Jaegyoon Ahn. 2022. Predicting chemical structure using reinforcement learning with a stack-augmented conditional variational autoencoder. *J. Cheminformatics* 14, 1 (2022), 83.

- [34] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems* 29 (2016).
- [35] Minghuan Liu, Menghui Zhu, and Weinan Zhang. 2022. Goal-conditioned reinforcement learning: Problems and solutions. *arXiv preprint arXiv:2201.08299* (2022).
- [36] Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. 2019. SDRL: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 2970–2977.
- [37] George H Mealy. 1955. A method for synthesizing sequential circuits. *The Bell System Technical Journal* 34, 5 (1955), 1045–1079.
- [38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.
- [39] Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. 2023. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning* 16, 1 (2023), 1–118.
- [40] Cyrus Neary, Zhe Xu, Bo Wu, and Ufuk Topcu. 2021. Reward Machines for Cooperative Multi-Agent Reinforcement Learning. In *Proceedings of the 2021 International Conference on Autonomous Agents and Multiagent Systems*. 934–942.
- [41] Daniel Neider and Nils Jansen. 2013. Regular model checking using solver technologies and automata learning. In *NASA Formal Methods Symposium*. Springer, 16–31.
- [42] José Oncina and Pedro García. 1992. Inferring regular languages in polynomial updated time. In *Pattern recognition and image analysis: selected papers from the IVth Spanish Symposium*. World Scientific, 49–61.
- [43] Ronald Parr and Stuart Russell. 1997. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems* 10 (1997).
- [44] Shubham Patelia, Budhitama Subagja, Ah-hwee Tan, and Chai Quek. 2021. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)* 54, 5 (2021), 1–35.
- [45] Amir Pnueli. 1977. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science, Providence*. 46–57.
- [46] Erika Puiutta and Eric MSP Veith. 2020. Explainable reinforcement learning: A survey. In *International cross-domain conference for machine learning and knowledge extraction*. 77–95.
- [47] Haoyuan Sun and Feng Wu. 2023. Less Is More: Refining Datasets for Offline Reinforcement Learning with Reward Machines. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. 1239–1247.
- [48] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
- [49] Richard S Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112, 1-2 (1999), 181–211.
- [50] Luca A. Thiede, Mario Krenn, AkshatKumar Nigam, and Alán Aspuru-Guzik. 2022. Curiosity in exploring chemical spaces: intrinsic rewards for molecular reinforcement learning. *Mach. Learn. Sci. Technol.* 3, 3 (2022), 35008.
- [51] Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. 2019. Learning reward machines for partially observable reinforcement learning. *Advances in neural information processing systems* 32 (2019).
- [52] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8 (1992), 279–292.
- [53] Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu, and Bo Wu. 2020. Joint Inference of Reward Machines and Policies for Reinforcement Learning. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling*. 590–598.
- [54] Chao Yu, Xuejing Zheng, Hankz Hankui Zhuo, Hai Wan, and Weilin Luo. 2023. Reinforcement Learning with Knowledge Representation and Reasoning: A Brief Survey. *CoRR* abs/2304.12090 (2023).
- [55] Mehdi Zadem, Sergio Mover, et al. 2023. Goal space abstraction in hierarchical reinforcement learning via set-based reachability analysis. In *2023 IEEE International Conference on Development and Learning (ICDL)*. IEEE, 423–428.
- [56] Mohammad Zangooei, Niloy Saha, Morteza Golkarifard, and Raouf Boutaba. 2023. Reinforcement Learning for Radio Resource Management in RAN Slicing: A Survey. *IEEE Commun. Mag.* 61, 2 (2023), 118–124.