Hierarchical Reinforcement Learning: A Comprehensive Survey

SHUBHAM PATERIA, Nanyang Technological University
BUDHITAMA SUBAGDJA and AH-HWEE TAN, Singapore Management University
CHAI QUEK, Nanyang Technological University

Hierarchical Reinforcement Learning (HRL) enables autonomous decomposition of challenging long-horizon decision-making tasks into simpler subtasks. During the past years, the landscape of HRL research has grown profoundly, resulting in copious approaches. A comprehensive overview of this vast landscape is necessary to study HRL in an organized manner. We provide a survey of the diverse HRL approaches concerning the challenges of learning hierarchical policies, subtask discovery, transfer learning, and multi-agent learning using HRL. The survey is presented according to a novel taxonomy of the approaches. Based on the survey, a set of important open problems is proposed to motivate the future research in HRL. Furthermore, we outline a few suitable task domains for evaluating the HRL approaches and a few interesting examples of the practical applications of HRL in the Supplementary Material.

CCS Concepts: • **Computing methodologies** → **Reinforcement learning**;

Additional Key Words and Phrases: Hierarchical reinforcement learning, subtask discovery, skill discovery, hierarchical reinforcement learning survey, hierarchical reinforcement learning taxonomy

ACM Reference format:
Shubham Pateria, Budhitama Subagdja, Ah-Hwee Tan, and Chai Quek. 2021. Hierarchical Reinforcement Learning: A Comprehensive Survey. *ACM Comput. Surv.* 54, 5, Article 109 (May 2021), 35 pages.
https://doi.org/10.1145/3453160

1 INTRODUCTION

One of the cardinal goals of Artificial Intelligence is to develop autonomous agents that can perform various complex tasks in an environment by planning the optimal sequences of actions [80]. **Reinforcement Learning (RL)** is a computational paradigm for *learning* a policy that takes optimal actions in various states of a task environment to maximize the cumulative rewards received by the acting agent [91]. To learn the optimal policy, the agent explores the state and action spaces relevant to the task by executing various sequences of *state-action-next state*

This research was supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-RP-2020-019) and the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier-1 grant (19-C220-SMU-023).
Authors' addresses: S. Pateria and C. Quek, School of Computer Science and Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore, 639798; emails: SHUBHAM007@e.ntu.edu.sg, ashcquek@ntu.edu.sg; B. Subagdja and A.-H. Tan, School of Computing and Information Systems, Singapore Management University, 80 Stamford Road, Singapore, 178902; emails: {budhitamas, ahtan}@smu.edu.sg.

0360-0300/2021/05-ART109 $15.00
https://doi.org/10.1145/3453160

transitions. The average length of such sequences[1] is called the task horizon. If the horizon is long while the task involves large state and action spaces, then the exploration space also becomes large. This results in the poor performance of the standard RL algorithms [56, 68, 100] on such *long-horizon tasks* without sophisticated exploration techniques [10, 71, 75].

Hierarchical Reinforcement Learning (HRL) decomposes a long-horizon reinforcement learning task into a hierarchy of subproblems or subtasks such that a higher-level policy learns to perform the task by choosing optimal subtasks as the higher-level actions. A subtask may itself be a reinforcement learning problem with a lower-level policy learning to solve it [39]. This hierarchy of policies collectively determines the behavior of the agent. Task decomposition effectively reduces the original task's long horizon into a shorter horizon in terms of the sequences of subtasks. This is because each subtask is a higher-level action that persists for a longer timescale compared to a lower-level action, a property that is often referred to as temporal abstraction [7, 20, 93]. Temporal abstraction can also enable efficient credit assignment over longer timescales [99]. At the same time, a subtask may itself be easier to learn and the learned subtasks lead to more structured exploration over the course of training of the HRL agent [71].

These aspects make HRL a promising approach to scale reinforcement learning to the long-horizon tasks [19, 69, 99]. HRL algorithms have been shown to outperform standard RL in several long-horizon problems such as continuous control[2] [25, 54, 69], long-horizon games [51, 99], robot manipulation, [28, 33] and so on. Different empirical studies find that the performance benefits of HRL are primarily due to improved exploration using subtasks/subgoals [43, 71].

HRL research has progressed significantly in the past three decades, resulting in a profusion of approaches that address a diversity of challenges such as learning the policies in a hierarchy, autonomous discovery of subtasks, transfer learning, and multi-agent learning using HRL. This causes significant difficulties in understanding the progress in the field in an organized manner. Hence, there is a necessity for a comprehensive survey to collect and organize the important HRL approaches and to provide a general taxonomy for their classification.

What are the differences between this survey and the previous ones? Barto et al [7] provided a survey of the advances in HRL up to the year 2003. That survey includes an important overview of the classical approaches, mainly MAXQ, Options [93], and HAMs [73]. Significant developments have occurred in the field of HRL since the time of that survey, such as subtask discovery using graph analysis, variational inference, autoencoding, unified HRL, subtask discovery in multi-agent HRL, transfer learning with HRL, and so on. Our survey mainly differs from that of Barto et al. [7] in the sense that we review the new HRL approaches that have emerged after their survey in addition to the classical approaches covered by them. Al-Emran et al. [21] performed a survey from the perspective of the practical applications of HRL. However, it does not include several important approaches that do not apply to the chosen applications, such as the recent unified HRL techniques, transfer learning with HRL, multi-agent HRL, and so on. In contrast, we survey the HRL approaches broadly and provide a general taxonomy that is application-agnostic.

The most recent survey by Mendonca et al. [67] gives a detailed review of the graph-based approaches for subtask discovery. Due to this limited scope, their survey goes into the depths of the graph-based subtask discovery but excludes other important aspects of HRL research such as the techniques for learning a hierarchy of policies, subtask discovery using variational inference, unified HRL, transfer learning with HRL, multi-agent HRL, and so on. However, we review all such approaches along with the graph-based subtask discovery (Section 3.3.1) in the spirit of providing a broader view of the HRL research.

[1]The length of a sequence is defined as the number of items in that sequence. In this case, an item is an action.
[2]An agent uses a continuous action space instead of a discrete set of actions.

Contributions of this survey. The objective of this survey is to thoroughly review the literature on HRL and provide a panorama of the approaches developed so far. The key contributions are as follows:

(1) We conduct a comprehensive survey of the work done so far in the field of HRL. The survey includes the approaches for learning hierarchies of policies, independent subtask discovery, unified HRL, multi-task/transfer learning with HRL, and multi-agent HRL.
(2) We provide a novel taxonomy to organize the HRL approaches along important characteristic dimensions such as single-agent vs. multi-agent, single-task vs. multi-task, and without subtask discovery vs. with subtask discovery.
(3) We identify a set of important open problems to provide the directions for future research concerning the scalability, efficiency, and theoretical robustness of HRL.

Our endeavor is to make this survey as comprehensive as possible with regard to the overview and classification of the HRL research in general, rather than the review of all the specific approaches in existence. The HRL literature is too vast to cover in a limited number of pages. Therefore, the main representative approaches for each class have been included in the survey while other similar approaches have been left out to conserve space.

The rest of the article is organized as follows: In Section 2, we review the general concepts of reinforcement learning, task decomposition, and HRL. Section 3 presents the taxonomy and survey of various HRL approaches along with the broad challenges addressed by them. Section 4 contains a discussion on the important open problems for future research on HRL. We conclude the survey in Section 5. In addition to this article, we provide a Supplementary Material document that outlines a few task domains for evaluating and benchmarking HRL approaches and contains a few interesting examples of the practical applications of HRL in the real-world problems.

2 PRELIMINARIES

2.1 Reinforcement Learning

RL refers to learning how to *take actions in different states of an environment* to *maximize cumulative future reward* [91], where the reward is a form of feedback associated with a task and received in response to an action. The majority of the current RL research is built upon the theory of **Markov Decision Processes (MDPs)** [12]. An MDP is control process defined using a tuple $< S, A, P, r >$. Here, S is the state space of the environment, where a state $s \in S$ represents a situation of the environment. A is the set of actions that can be taken by an agent. $r : S \times A \rightarrow \mathbb{R}$ is the numerical reward obtained as a function of state and action. The effect of an action on the future state is captured by a probabilistic transition function $P(s_{t+1}|s_t, a_t) \rightarrow [0, 1]$ where $s_{t+1} \in S$ is the next state observed after taking action $a_t \in A$ in a state $s_t \in S$ at time t. An MDP is stochastic if there exists a state-action pair (s_t, a_t) for which $P(s_{t+1}|s_t, a_t) \neq 1$, because the effect of an action is not deterministic. Contrarily, an MDP is deterministic if for every state-action pair (s_t, a_t), $P(s_{t+1}|s_t, a_t) = 1$. The agent takes actions according to a policy $\pi : S \times A \rightarrow [0, 1]$. This policy may be stochastic.

One step of interaction between the agent and the environment produces a transition from current state s_t to next state s_{t+1}, observed as an experience data sample (s_t, a_t, r_t, s_{t+1}). A sequence of such transitions constitutes a trajectory. The trajectory taken from any starting state and action pair (s_t, a_t) is affected by the stochasticity in the policy and in the state transitions (as per $P(s_{t+1}|s_t, a_t)$). Thus, the value of taking a_t in s_t is formally calculated as the *expected* (or average)

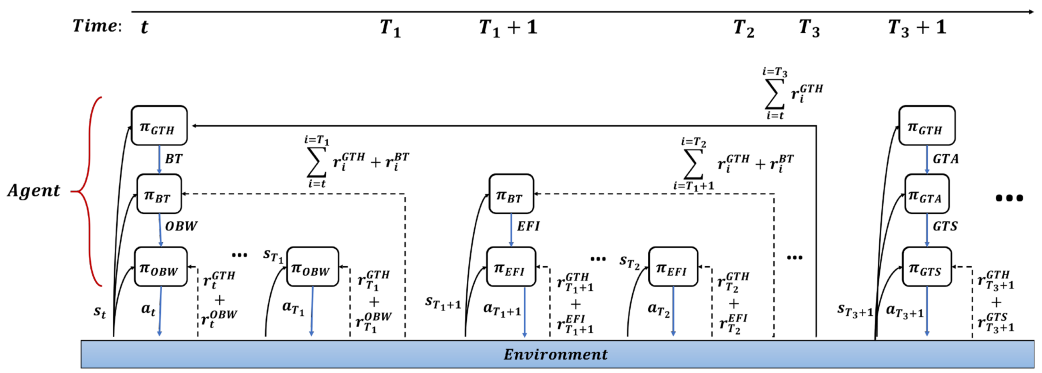

Fig. 1. **The temporal process of Hierarchical Reinforcement Learning.**

cumulative reward obtained over all possible trajectories, as follows:

$$Q^{\pi}(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{a\sim\pi(s)}\left[\sum_{i=1}^{\infty} \gamma^i r(s_{t+i}, a_{t+i}) | s_t, a_t\right]. \tag{1}$$

Here, $Q^{\pi}(s_t, a_t)$ is called the *Q-value* of the state-action pair while an agent follows a policy π. $\gamma \in [0, 1)$ is called the *discount factor*. $a \sim \pi(s)$ implies that an action is sampled using the policy π. **The core objective of RL** is to learn an optimal policy π^* that satisfies the following condition:

$$\pi^* = \arg\max_{\pi} Q^{\pi}(s, a), \forall s \in S, \forall a \in A. \tag{2}$$

2.2 Hierarchical Reinforcement Learning

It can be inferred from Equations (1) and (2) that the objective of an RL agent is to search for a policy that maximizes the cumulative reward averaged over various possible trajectories the agent can take while following that policy. While exploring the state and action spaces to learn the optimal policy, the agent takes various trajectories whose expected length is the task horizon (the horizon is shown as theoretically infinite in Equation (1), but in practice it may be finite). As mentioned in Section 1, when the state and action spaces are large, and the task horizon is long, the exploration becomes challenging using the standard RL approach [10, 71, 75].

HRL provides a mechanism to perform a challenging task by decomposing it into simpler subtasks using a hierarchy of policies learned via reinforcement learning. In such a hierarchy, the highest-level policy generally chooses the subtasks of the main task as its actions [39]. This policy is trained to perform the main task in terms of the sequence of its subtasks, using the rewards obtained in the main task. At a lower level in the hierarchy, a subtask chosen by the higher-level is itself a reinforcement learning problem. A lower-level policy learns to perform that subtask using the internal rewards related to it (optionally, the main task reward may also be added). The lowest-level policies choose the basic actions that are henceforth referred to as *primitive* actions.

The process of HRL is illustrated in Figure 1 using an example of task decomposition given by Russell et al. [80]. In this illustration, an HRL agent decomposes and performs a long-horizon task "Going to Hawaii" (GTH). The HRL agent is composed of a hierarchy of policies. The task policy π_{GTH} decomposes the original task GTH into a sequence of highest-level subtasks "Book Tickets" (BT), "Go To Airport" (GTA), and so on. The task policy initially chooses BT. Then, BT is executed for multiple timesteps until its termination occurs at time T_3. During this period, the policy of the subtask BT, i.e., π_{BT}, itself chooses different shorter subtasks sequentially. These are "Open Booking Website" (OBW), "Enter Flight Information" (EFI), and so on. After BT terminates at T_3,

the task policy chooses GTA, which itself chooses a shorter subtask "Go to Taxi Stand" (GTS). During each timestep, a primitive action a is chosen by the lowest level subtask policy, for, e.g., by π_{OBW}, π_{EFI}, π_{GTS}, and so on. The HRL agent receives a task reward r^{GTH} in response to the primitive action, which is accumulated and given at different time-scales to different levels. The task reward may be optional for the policies below the highest level, which can be trained using the subtask-related rewards. In this manner, the HRL process continues in time until GTH finishes.

2.2.1 Formal Definition of a Subtask. The definition of a subtask in the formal context of HRL is provided in this subsection. This definition is a culmination of the different ways in which different HRL approaches define the subtasks. Hence, it should be considered as a *general* definition. A particular HRL approach may only use selective components of the general definition. The specifics are further discussed in Section 3.

First, we denote the main long-horizon task as Γ and the task policy as π_Γ. The task policy is at the top of a hierarchy, for e.g., π_{GTH} in Figure 1. A subtask is denoted as ω. It is defined using the components described as follows:

- π_ω, which is the **policy of the subtask**. It maps the environment states to primitive actions or to the subtasks of ω [93].
- The **objective components**:
  - r_ω, which is the subtask reward used to train π_ω. This is typically different from the reward associated with the main task [51, 69, 99],
  - g_ω, which is a subgoal or a set of subgoals associated with ω. A subgoal might be a state $s \in S$ itself [64, 84, 87], an abstract form of a state [54, 69], a learned embedding [70, 88, 99], and so on. The reward r_ω might be defined with respect to the subgoal(s).
- The **execution components**:
  - I_ω, which is the initiation condition of ω. It may be defined as a set of states in which ω can be chosen for execution [93], as a function that modulates the probability of choosing ω in a particular state [45], as a set of logical conditions [74], and so on.
  - β_ω, which is the termination condition of ω. It may be defined as a set of states in which ω should terminate if it is being executed [19, 51], as a function that modulates the probability of terminating ω in a particular state [3, 93], as a fixed time-limit [54, 69], and so on. If g_ω is defined, then the subgoal state is also usually included as a state in which ω must terminate [51].

In the remainder of this article, the term "subtask" should be inferred as referring to this formal definition.

2.2.2 Formalism of HRL Based on Semi-Markov Decision Process. HRL is formalized on the basis of the theory of **Semi-Markov Decision Process (SMDP)** [9]. An SMDP is a stochastic control process similar to an MDP (Section 2.1), but unlike MDP, it also involves the concept of *time* for which an action is executed after it has been chosen. In the context of HRL, the actions with the concept of time are the *subtasks*. Starting from a state $s_t \in S$, assume that an agent chooses a subtask $\omega_t \in \Omega$, where Ω is the set of subtasks (or the subtask space). Then, the transition function of the SMDP is defined as a joint distribution

$$P(s_{t+c_{\omega_t}}, c_{\omega_t} | s_t, \omega_t) = P(s_{t+c_{\omega_t}} | s_t, \omega_t, c_{\omega_t}) P(c_{\omega_t} | s_t, \omega_t). \tag{3}$$

Here, c_{ω_t} denotes the number of timesteps for which ω_t is executed, starting from the state s_t. c_{ω_t} is actually determined by the termination condition β_{ω_t}, which is one of the execution components defined in Section 2.2.1.

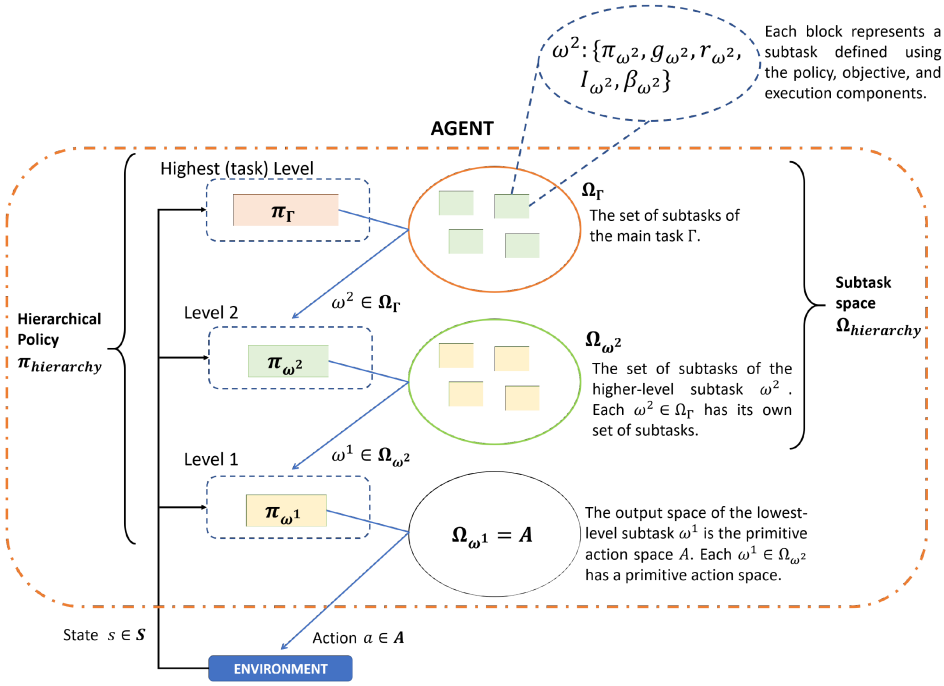

Fig. 2. **The concept of a Hierarchical Reinforcement Learning agent.** This illustration is for a three-level hierarchy, but the concept can be extrapolated to more than three levels.

The reward obtained in response to performing the subtask ω_t starting from state s_t is denoted as $R(s_t, \omega_t)$, calculated as follows,

$$R(s_t, \omega_t) = \mathbb{E}_{a \sim \pi_{\omega_t}(s)} \left[\sum_{i=0}^{c_{\omega_t}-1} \gamma^i r(s_{t+i}, a_{t+i}) | s_t, a_t = \pi_{\omega_t}(s_t) \right]. \tag{4}$$

Equation (4) indicates that the reward $R(s_t, \omega_t)$ is equal to the expected cumulative reward obtained while following the subtask policy π_{ω_t} from time t until the termination of ω_t after c_{ω_t} timesteps. Now, an optimal task policy would be the one that leads to the following desired maximum Q-value:

$$Q(s_t, \omega_t) = R(s_t, \omega_t) + \sum_{s_{t+c_{\omega_t}}, c_{\omega_t}} \gamma^{c_{\omega_t}} P(s_{t+c_{\omega_t}}, c_{\omega_t} | s_t, \omega_t) \max_{\omega_{t+c_{\omega_t}}} Q(s_{t+c_{\omega_t}}, \omega_{t+c_{\omega_t}}), \tag{5}$$

$\forall s \in S$ and $\forall \omega \in \Omega$.

It should be noted that the Q-value in Equation (5) also depends on $R(s_t, \omega_t)$ and $P(s_{t+c_{\omega_t}}, c_{\omega_t} | s_t, \omega_t)$. These two quantities are determined by the execution of ω_t using its policy π_{ω_t}. Therefore, an agent actually needs to learn multiple policies at different levels of a task decomposition hierarchy, including π_Γ and the policies of all the subtasks (e.g., Figure 1). We extend the notations of subtasks and policies for a multi-level hierarchy as defined below. Please refer to Figure 2 for an example of these notations in the context of an HRL agent with a three-level hierarchy.

- ω^l: a subtask at level l of the hierarchy,
- Ω_{ω^l}: set of subtasks under the subtask ω^l such that $\omega^{l-1} \in \Omega_{\omega^l}$,

- $\pi_{\omega^l}: S \times \Omega_{\omega^l} \rightarrow [0, 1]$: policy of the subtask ω^l. In other words, ω^{l-1} is chosen by π_{ω^l},
- $\Omega_{\omega^1} = A$, i.e., the output space of a subtask at the lowest level ($l = 1$) is the primitive action space A,
- π_Γ and Ω_Γ denote the main task policy and the set of subtasks at the highest level, respectively.

Consolidating all the definitions provided above, there are two principal parts of an HRL agent:

- **Subtask space** ($\Omega_{hierarchy}$). This is the super-set of all the subtasks used in a hierarchy, i.e., $\Omega_{hierarchy} = \{\Omega_{\omega^2}, \Omega_{\omega^3}, \Omega_{\omega^4}, \ldots, \Omega_\Gamma\}$.
- **Hierarchical policy** ($\pi_{hierarchy}$). The primitive action taken by the HRL agent is the result of the recursive choices of subtasks. Consider the three-level hierarchy depicted in Figure 2. In this hierarchy, the main policy π_Γ chooses a level 2 subtask, i.e., $\omega^2 = \pi_\Gamma(s)$ where $\omega^2 \in \Omega_\Gamma$. The policy of ω^2 is executed until its termination as per β_{ω^2}. It chooses the lowest-level subtask $\omega^1 = \pi_{\omega^2}(s)$ where $\omega^1 \in \Omega_{\omega^2}$. The policy of ω^1 is executed until its termination as per β_{ω^1}. This lowest-level policy selects a primitive action, i.e., $a = \pi_{\omega^1}(s)$. This complete *state-to-subtask-to-action mapping* from π_{ω^1} is called the *hierarchical policy*, denoted as $\pi_{hierarchy}$. Now, the primitive action taken by the HRL agent *as a whole* can be equivalently expressed as $a = \pi_{hierarchy}(s)$. This description can be extrapolated to the hierarchies with more than three levels.

Based on the definitions provided above, the expected discounted cumulative reward received by the HRL agent can be written as,

$$Q^{hierarchy}(s_t, a_t) = \mathbb{E}_{a \sim \pi_{hierarchy} | \Omega_{hierarchy}} \left[\sum_{i=0}^{\infty} \gamma^{t+i} r(s_{t+i}, a_{t+i}) | s_t, a_t \right], \tag{6}$$

where $a \sim \pi_{hierarchy} | \Omega_{hierarchy}$ indicates that a primitive action a is sampled using the hierarchical policy $\pi_{hierarchy}$ conditioned on the available subtask space $\Omega_{hierarchy}$.

2.2.3 Problem Definition of HRL. The general problem definition of HRL is to find the optimal hierarchical policy $\pi^*_{hierarchy}$ and the optimal subtask space $\Omega^*_{hierarchy}$ as the solution to:

$$\Omega^*_{hierarchy}, \pi^*_{hierarchy} = argmax_{\Omega_{hierarchy}} argmax_{\pi_{hierarchy} | \Omega_{hierarchy}} Q^{hierarchy}(s, a), \forall s \in S, a \in A. \tag{7}$$

The HRL problem expressed in Equation (7) can be divided into two parts. The first part is ***learning hierarchical policy***, which refers to finding the optimal hierarchical policy conditioned on the available subtask space (i.e., $argmax_{\pi_{hierarchy} | \Omega_{hierarchy}}$). This is essential, because the hierarchical policy determines the behavior of an HRL agent on any given task. The policies at various levels of $\pi_{hierarchy}$ can be learned simultaneously in an end-to-end manner [3, 21, 52, 54, 69] or they may be learned one level at a time in a bottom-to-top manner [25, 28, 60]. The second part is ***subtask discovery***, which refers to automatically finding the optimal subtask space using the HRL agent's experience data (i.e., $argmax_{\Omega_{hierarchy}}$). Subtask discovery is not essential, because a subtask space could be handcrafted using precise domain knowledge. However, it is necessary for generalized HRL without dependence on the manual handcrafting.

2.3 Definitions of Common Terms and Concepts

A few important terms and concepts that appear commonly in the rest of this article are defined here.

Skill. A "skill" is a semantic term for referring to a primitive action policy learnt to perform a subtask, in the sense that such a policy represents the *ability of an agent to do something well.*

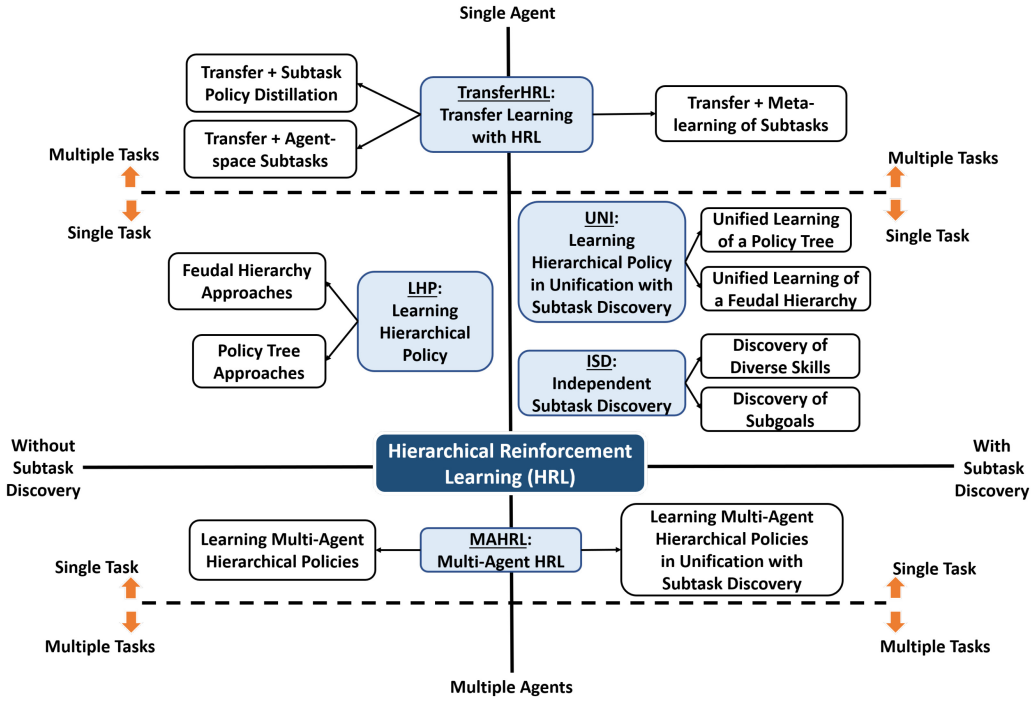

Fig. 3. **A taxonomy of HRL approaches**. The approaches are arranged along the following three dimensions: with or without subtask discovery, for single agent or multiple agents, and for single task or multiple tasks.

Universal policy. In this article, a "universal policy" means a policy that can learn all the possible subtasks by taking the corresponding subgoals [19, 54, 99] or instructions [41] as input, i.e., $\pi_\omega(s) = \pi(s, g_\omega)$, where $\pi(s, g_\omega)$ is the universal policy.

State abstraction. A state abstraction is a mapping, say, ϕ, that maps the original state space S to some finite lower-dimensional abstract space. If $\phi(s) = \phi(s')$ for a pair $s, s' \in S$, then the two states in the original state space are represented by the same state in the abstract space.

Reward abstraction. A reward abstraction means that the rewards given as part of the main task are hidden from the subtask policies and only received by the highest-level policy.

"Global" initiation condition. In the case of *definition of a subtask* in the tables provided in Section 3, wherever the term "global" is used in reference to the initiation condition I_ω, it means that the subtask can be chosen for execution in any of the states in the global state space.

3 APPROACHES FOR HIERARCHICAL REINFORCEMENT LEARNING

This section presents the review and classification of a wide range of approaches developed for Hierarchical Reinforcement Learning. The review is structured according to a novel taxonomy depicted in Figure 3. To design this taxonomy, the surveyed HRL approaches are arranged along three independent dimensions, which are as follows: (1) Approaches with subtask discovery or without it. (2) Approaches for training single agent or multiple agents. (3) Approaches for learning on single task or multiple tasks. There can be eight possible divisions (octants) of this three-dimensional space. We identify that the majority of the HRL approaches lie in six of these divisions. The approaches are further grouped into five major classes—LHP, UNI, ISD, MAHRL, and TransferHRL—based on the broad challenge addressed by them. The divisions and classes are as follows:

(1) *Single agent, single task, without subtask discovery*: The approaches in this division are grouped into a major class called *Learning Hierarchical Policy* **(LHP)**. The LHP approaches address the challenge of learning the hierarchical policy $\pi_{hierarchy}$ of an HRL agent without concerning with subtask discovery. They use handcrafted subtasks.
(2) *Single agent, single task, with subtask discovery*: The approaches in this division are grouped into two major classes. The first class is called *Learning Hierarchical Policy in Unification with Subtask Discovery* **(UNI)**. The UNI approaches address the challenge of learning the subtask space $\Omega_{hierarchy}$ and the hierarchical policy $\pi_{hierarchy}$ in a unified or end-to-end manner. The second class is called *Independent Subtask Discovery* **(ISD)**. The approaches in this class address the challenge of discovering task-agnostic subtasks independently from any specific task. The subtask discovery is usually performed in a pre-training stage and then the subtasks are used to learn HRL agents on downstream tasks.
(3) *Multiple agent, single task, without subtask discovery.*
(4) *Multiple agent, single task, with subtask discovery.* The approaches in divisions 3 and 4 are grouped into one class called *Multi-Agent HRL* **(MAHRL)**. The MAHRL approaches broadly address the challenge of learning to coordinate among multiple HRL agents on a single joint task.
(5) *Single agent, multiple tasks, without subtask discovery.*
(6) *Single agent, multiple tasks, with subtask discovery.* The approaches in divisions 5 and 6 are grouped into one class called *Transfer learning with HRL* **(TransferHRL)**. The TransferHRL approaches broadly address the challenge of learning to transfer the hierarchical policy, subtasks, or other knowledge of an HRL agent across multiple tasks, where the subtasks may be handcrafted or discovered from scratch on multiple tasks.

We could not find considerable work in the remaining two divisions that represent the approaches for *multiple agents, multiple tasks, with/without subtask discovery*. A detailed discussion on the above-mentioned major classes, their associated challenges, and the approaches that address those challenges is provided in the following subsections. Before proceeding, we encourage the reader to recall the important definitions of the common terms provided in Section 2.3.

3.1 Learning Hierarchical Policy (LHP)

This subsection reviews the approaches that address the challenge of learning the hierarchical policy $\pi_{hierarchy}$, without subtask discovery, for single agent and single task setting. Learning the hierarchical policy is a non-trivial challenge even when the subtasks are handcrafted. This is because any approach to learn a hierarchical policy must tackle the following key issues: carefully designing the algorithm to learn the policies at various levels of the hierarchy (including reward propagation, value function decomposition, state/action space design, etc.), dealing with non-stationarity due to simultaneously changing policies, ensuring the optimality of the hierarchical policy as a whole, interpretability of the hierarchical policy, among other issues. Please refer to Table 1 for a summary of the LHP approaches.

The LHP approaches can roughly be put into two sub-classes. The first sub-class is called *feudal hierarchy*, in which the action space of a higher-level policy consists of subgoals corresponding to various subtasks. A subgoal chosen by the higher-level policy is taken as input by a universal policy[3] at the lower level. The objective of this lower-level universal policy is to achieve the input subgoal. The universal policy at each level can be treated as a sub-agent (as part of the HRL

[3]Defined in Section 2.3.

Table 1. Approaches for Learning Hierarchical Policy (LHP)

Approach	Definition of a subtask ω					State abstraction	Reward abstraction	Suitable task domains		Main utilities	Main limitations
	I_ω	π_ω	β_ω	g_ω	r_ω			state space	action space		
Feudal Hierarchy Approaches											
Feudal RL; Dayan & Hinton [19]	global	learned using r_ω	fixed timesteps or at g_ω	PRE	w.r.t. g_ω	Possible	Yes	D, S	D	learns hierarchical policy using subgoals	hierarchical policy is likely to be sub-optimal // does not address the issue of non-stationarity
Deep HRL; Kulkarni et al. [51]	global	learned using r_ω	fixed timesteps or at g_ω	PRE	w.r.t. g_ω	Possible	Yes	D/C, S/L	D	learns hierarchical policy using subgoals in the tasks with large and high-dimensional state spaces	hierarchical policy is likely to be sub-optimal // does not address the issue of non-stationarity
HIRO; Nachum et al. [69] // *HAC*; Levy et al. [54]	global	learned using r_ω	fixed timesteps or at g_ω	PRE, using selective features of the state space	w.r.t. g_ω	Possible	Yes	C, L/S	C	reduce non-stationarity while learning a feudal hierarchy	manually mapping the state space to the subgoal space may be challenging if the state space is high-dimensional
HAL Jiang et al. [41]	global	learned using r_ω	fixed timesteps or at g_ω	PRE, using natural language instructions	w.r.t. g_ω	Possible	Yes	D/C, S/L	C	interpretability of hierarchical policy using natural language	requires predefined mapping of language instructions to the task domain
Policy Tree Approaches											
Options; Sutton et al. [93]	PRE	PRE, further tuned using task rewards	PRE	n.r.	optional	No	No	D, S	D	learns optimal hierarchical policies	all the Options are blended into the same MDP, so Options are not easily transferable
MAXQ; Dietterich	global	learned using r_ω	PRE	n.r.	PRE	Possible	Yes	D, S	D	policies of the subtasks are transferable across different tasks/hierarchies	hierarchical policy is likely to be sub-optimal
HAMs; Parr & Russell [73]	PRE	learned using task rewards	PRE	n.r.	optional	No	No	D, S	D	prior knowledge about higher-level transition dynamics can be added by using Finite State Machines (FSMs) [13]	complicated to implement due to the intricate programming of FSMs

w.r.t. = *with respect to*; n.r. = *not required.*
PRE = predefined, D = discrete, C = continuous, S = small, L = large.
All the task domains are assumed to have moderate to extreme reward sparsity.

agent) that can perform all the possible subtasks at that level. This leads to the *feudal* concept of a "manager" sub-agent (higher-level policy) directing a "worker" sub-agent (lower-level policy) [19, 51, 69]. The corresponding approaches are discussed in Section 3.1.1.

The second sub-class is called *policy tree*. This name is introduced for the purpose of classification in this survey, and it may not be found in the HRL literature. In a policy tree, the action space of a higher-level policy consists of the *different* lower-level policies of subtasks. The subtasks are not represented by a single universal lower-level policy; rather, each subtask has a separate policy. The higher-level policy and the various lower-level policies form a ''tree.'' The higher-level policy selects a lower-level policy directly for execution without intermediate subgoals [73, 93]. The corresponding approaches are discussed in Section 3.1.2.

3.1.1 Feudal Hierarchy Approaches. Dayan and Hinton [19] introduced the foundational feudal hierarchy called **Feudal Reinforcement Learning (Feudal RL)**. Feudal RL is briefly described as follows: a higher-level manager sets a subtask that is to be executed by a lower-level worker. This is a pairwise relation such that the worker becomes the manager of the level below it if the hierarchy has more than two levels. The manager communicates the subtask to the worker via a subgoal, where a subgoal is simply a state in the original or abstract state space. The objective of the worker is to reach the given subgoal. The worker at each level is a universal policy. The task reward may only be observed by the manager at the highest level while the workers at other levels learn using the rewards for reaching the subgoals. The authors evaluate Feudal RL on a maze navigation task in a grid-world environment using predefined states at different spatial distances as subgoals. Feudal RL converges to a shorter path to a main goal state in the maze faster than the standard Q-learning. Rest of the feudal hierarchy approaches discussed in this article are based on the concept of Feudal RL described above.

Kulkarni et al. [51] proposed a *Deep HRL* approach consisting of a two-level hierarchy of **Deep Q Networks (DQNs)** [68] that represent the manager and the worker. The manager network selects a subgoal from a set of predefined subgoals. It is learned using the task reward. The worker network takes the subgoal as input and selects primitive actions to achieve the subgoal. It is learned using rewards for reaching the subgoal. A subgoal is either an original state or an abstract representation of a state, for e.g., an object extracted from a image where the image represents the original state. One of the tasks used for evaluation is Atari Montezuma's Revenge, in which standard DQN makes almost no progress in terms of collected rewards over 2M timesteps. However, Deep HRL with predefined subgoals progressively gets higher rewards as training proceeds.

Learning multiple levels of policies simultaneously results in **the issue of non-stationarity** [54, 69]. This means that the *state-action-next state* transition data, which is generated by executing a lower-level policy and is observed by a higher-level policy, varies over different time instances even for the same subgoal chosen in the same state of the environment. This is because the lower-level policy is not stationary and its response to a *(state, subgoal)* pair changes during the learning process. Non-stationarity may result in many useless data samples, and it needs to be addressed for data-efficient learning.

To address the issue of non-stationarity, Nachum et al. [69] proposed a two-level feudal hierarchy with a mechanism for *subgoal re-labelling*. This approach is called **Hierarchical Reinforcement Learning with Off-policy Correction (HIRO)**. Subgoal re-labelling in HIRO can be described as follows: A two-level HRL agent interacts with its environment and gathers experience data. This data consists of transition tuples $(s_t, g_t, \sum r_{t:t+c}, s_{t+c})$ for the higher-level policy where r is the task reward, and $(s_{t+i}, g_t, a_{t+i}, r^{g_t}_{t+i}, s_{t+i+1}), \forall\, 0 < i < c$, for the lower-level policy. Here, c is a fixed time horizon for achieving each subgoal and r^{g_t} is the reward for achieving g_t. This data

is subsequently used for training the HRL agent's hierarchical policy. However, if the agent does not achieve g_t after c timesteps, then the subgoal is re-labelled in the transition data with another subgoal g'_t drawn from a distribution of subgoals that maximize the probability of the observed transitions. Then, the higher-level policy treats g'_t as its output in the hindsight, which correlates better with the observed transitions. In this way, HIRO reduces the effective non-stationarity. A subgoal in HIRO is defined by choosing selective features from a state in the original state space. HIRO is evaluated on MuJoCo [97] continuous control tasks and shown to perform better than standard RL and few other HRL approaches [25, 99].

Concurrently with HIRO, Levy et al. [54] proposed an approach called the **Hierarchical Actor-Critic (HAC)**, which also addresses the non-stationarity issue by subgoal re-labelling. In this scheme, the output subgoal in the higher-level data and the input subgoal in the lower-level data are replaced with the actual state achieved by the agent in hindsight rather than a probability-based sampling of a new subgoal as in HIRO. This simple scheme also makes it possible to extend the hierarchy to more than two levels. HAC is evaluated on MuJoCo [97] continuous control tasks and shown to perform better than standard RL and HIRO. Moreover, the authors find that a three-level hierarchy performs better than a two-level hierarchy on those tasks.

A feudal hierarchy can also be realized by using *natural language instructions* instead of subgoal states. Jiang et al. [41] proposed an approach called **Hierarchical Abstraction with Language (HAL)**. They assume that a mapping from a state $s \in S$ to a language instruction[4] $l \in L$ is given by a human supervisor or generated using a predefined program, as a probability distribution over the instructions in the instruction space L. On this basis, a Boolean function $\Psi : S \times L \rightarrow (0, 1)$ is defined. Now, if a higher-level policy selects an instruction l, then all the states for which $\Psi(s, l) = 1$ are the subgoal states for that instruction. In terms of the notation used in this article, a language instruction l is a subtask ω and all the states that satisfy $\Psi(s, l) = 1$ belong to the subgoal set of ω, i.e., g_ω. The language instruction is encoded using a recurrent neural network [16] whose output is given to the lower-level policy as input. The subtask reward r_ω used to train the lower-level policy is a binary reward for achieving the states that satisfy the given instruction. HAL also handles non-stationarity using a mechanism called Hindsight Instruction Re-labelling. HAL is evaluated on object arrangement tasks in MuJoCo continuous control domain [100] integrated with CLEVR language dataset [42] and found to outperform standard RL and few other HRL approaches [3, 69].

3.1.2 Policy Tree Approaches. Sutton et al. [93] introduced the Options framework in which the default MDP[5] (called *core* MDP) is extended by including a set of subtasks, called *Options*, into the action space of an agent. A primitive action itself is considered as a single-step Option. An Option ω is defined using a tuple: $(< I_\omega, \pi_\omega, \beta_\omega >)$. $I_\omega \subset S$ is a set of states in which the Option ω can be initiated or is valid. β_ω: $S \rightarrow [0, 1]$ gives the probability of termination of ω in a state $s \in S$. π_ω: $S \times A \rightarrow [0, 1]$ is the policy of the Option. An Option *may or may not* have an associated reward function r_ω. It is usually assumed that the Options are predefined by a programmer. The policy of an Option (π_ω) is predefined as an initial policy that can be fine-tuned in the context of a particular task using a mechanism called intra-option learning. Intra-option learning uses the task rewards even if the Option-specific reward r_ω is given. Moreover, the Q-value function of any Option ω, i.e., $Q_\omega(s, a)$, represents the expected cumulative reward until the end of the main task rather than only until the horizon (or termination) of the Option itself. Hence, an Option is not a standalone subtask unit but it *blends* into the core MDP. This limits the transfer-ability of the learned Options to other tasks but theoretically guarantees the optimality of the learned hierarchical policy. Sutton

[4]We use different notations compared to the original HAL paper [41] to avoid conflicts with the subtask notations (Section 2.2.1).
[5]Markov Decision Process, defined in Section 2.1.

et al. [93] evaluated the Options framework on grid-world tasks and found that it converges to optimal performance faster than standard RL.

MAXQ value function decomposition, proposed by Dietterich, decomposes the core MDP into smaller sub-MDP components. Each sub-MDP is associated to a subtask whose policy can be learned separately from other subtasks. This is achieved by decomposing the main Q-value function into the separate Q-value functions of the subtasks. Due to such decomposition, the Q-value of any subtask ω, i.e., $Q_\omega(s, a)$, represents the expected cumulative reward only until the horizon (or termination) of that subtask. Thus, each subtask policy can be learned as a standalone unit. This is in contrast against Options [93] in which a subtask is blended into the core MDP. The benefit of value decomposition is that the policies of the subtasks learned using MAXQ on one task can be easily transferred to different tasks or different MAXQ agents. The disadvantage of treating subtasks as standalone units is that the optimality of the hierarchical policy is compromised. MAXQ hierarchies are only *recursively* optimal, which means that MAXQ can learn optimal policies of the subtasks, but the overall hierarchical policy is generally not optimal. In MAXQ, the predefined components of a subtask ω are its termination condition β_ω (e.g., events, conditions, subgoal states) and reward function r_ω. A subtask may choose primitive actions in addition to choosing other subtasks. MAXQ is evaluated in a grid-world Taxi domain where it outperforms standard Q-learning.

Hierarchy of Abstract Machines (HAMs), by Parr and Russell [73], uses stochastic **Finite State Machines (FSMs)** [13] to represent the subtasks. HAMs is defined by a collection of stochastic FSMs $\{\mathbb{H}\}$. Each FSM $\mathbb{H}_i \in \{\mathbb{H}\}$ is defined using various machine-states. A machine-state is different from the state of the environment. An FSM consists of four *types* of machine-states: *action, call, choice,* and *stop*. There may be multiple machine-states of the same type. E.g., Parr and Russell [73] illustrate an FSM with two *call* machine-states - "Follow Wall" and "Back Off," —which can be chosen from the *choice* machine-state of that FSM. The *choice* machine-state non-deterministically selects the next machine-state of $\mathbb{H}_i$. The possible choices consist of the various *action*, *call*, or *stop* machine-states of $\mathbb{H}_i$. A Q-table captures the values of the *(state, choice)* pairs. The choices are made using a greedy policy. Thus, reinforcement learning (specifically, Q-learning) is used to learn the optimal choice to be made in the *choice* state, given the current state of the environment. If an *action* machine-state is chosen, then it executes a primitive action in the environment. If a *call* machine-state is chosen, then it suspends the execution of $\mathbb{H}_i$ and initiates the execution of another FSM, say, $\mathbb{H}_j$. This is equivalent to a transition from one subtask to another. If a *stop* machine-state is chosen, then it terminates the execution of $\mathbb{H}_i$ and returns the control to the FSM that called $\mathbb{H}_i$. HAMs is a useful and elaborate framework for programming the prior knowledge about the transition dynamics in a domain. However, due to its intricate structure, HAMs has not found widespread application in HRL.

3.2 Learning Hierarchical Policy in Unification with Subtask Discovery (UNI)

This subsection reviews the approaches that unify subtask discovery with the learning of the hierarchical policy for single agent and single task setting. For an HRL agent to be deployable on a particular task without requiring predefined or handcrafted subtasks, it is essential that the subtask space ($\Omega_{hierarchy}$) is discovered *simultaneously* when the hierarchical policy of the agent is being learned. This requires a unified or end-to-end learning approach that uses the same data for both subtask discovery and learning hierarchical policy, where the data is collected while performing the given task. A few important issues involved with such a unification are as follows: discovering the subtask space that maintains the optimality of the hierarchical policy, learning various components of a subtask during discovery (terminations conditions, initiation conditions, subgoals, etc.) from scratch, discovering a dynamic number of subtasks, among other issues.

The UNI approaches discussed henceforth are based on the two types of LHP approaches—the feudal hierarchy (Section 3.1.1) and Options (a policy tree approach, Section 3.1.2). Please refer to Table 2 for a summary of the approaches.

3.2.1 Unified Learning of Policy Tree. The policy tree approaches discussed in this subsection discover Options [93] in unification with the learning of a hierarchical policy. As discussed in Section 3.1.2, an Option is a subtask that is represented using three key components: initiation condition I_ω, policy π_ω, and a termination probability function β_ω. Unified Option discovery mainly involves the learning of π_ω and β_ω associated with each option ω simultaneously with the higher-level task policy (π_Γ). The initiation I_ω may also be learned or set as global.[6] None of these components are handcrafted.

Daniel et al. [17] propose a graphical model for unified Option discovery. At any timestep t, the observed variables of this graphical model are the environment state s_t and the primitive action a_t. The hidden variables are the Option ω_t active at that timestep and the termination label b_t (binary, 0 or 1) indicating whether the previous Option ω_{t-1} terminates (b_t=1) or not (b_t=0). The model consists of the activation policy $\mu(\omega|s;\theta_O)$, which is parameterized using θ_O and provides the probability distribution over all the Options, conditioned on the current state s_t. Each Option ω consists of a sub-policy $\pi(a|s;\theta_A^\omega)$ and a termination function $\beta(b|s;\theta_B^\omega)$. The set of θ_A^ω parameters of all the Options is denoted as θ_A. Similarly, the set of θ_B^ω parameters of all the Options is denoted as θ_B. Then, the parameters $\{\theta_O, \theta_A, \theta_B\}$ are learned using **Expectation-Maximization (EM)** [8] with weighted estimates in the maximization step. The EM algorithm takes a set of trajectories as input and learns the parameters that best explain the trajectories. The trajectories are obtained from the exploration done by the HRL agent. In the maximization step, each state-action (s, a) pair is weighted in proportion to the advantage value $Q(s, a) - V(s)$. In this way, the Option learning process incorporates the expected task reward. Overall, the parameters $\{\theta_O, \theta_A, \theta_B\}$ for the activation policy, all the sub-policies, and all the termination functions are learned in conjunction during EM. Hence, this is a unified HRL approach. This approach is evaluated on grid-world navigation task with discrete state-action space and pendulum swing task with continuous state-action space. It is found to perform better than standard RL and few other subtask discovery approaches [66, 85]. The main limitation of this approach is that the number of Options needs to be predefined and fixed.

Konidaris et al. [50] propose an approach called *skill chaining* to incrementally construct Options while learning an HRL agent. The process of skill chaining starts with one Option ω created with its subgoal set (g_ω) containing the goal states of the main task. The Option policy π_ω is learned to reach the subgoals. A classifier is then learned to find the initiation states of ω. The states from which the subgoals of ω can be reached within a predefined number of timesteps are classified as positive. In contrast, those states from which the subgoals cannot be reached within the time limit are classified as negative. The positive states are added to the initiation set I_ω. Now, another Option ω' is created, whose *subgoal set $g_{\omega'}$ is equal to I_ω*. Then, the above-mentioned classification and policy learning processes repeat for ω', and so on, creating a chain of Options (or skills). Skill chaining discovers a flexible number of Options. The authors evaluate it on a Pinball control domain in which it outperforms standard RL. Recently, Bagaria et al. [4] extended skill chaining for continuous control using deep reinforcement learning. Key limitations of skill chaining are that it works only on the tasks that have explicit goal states and requires strong initial exploration without Options to reach the goal states.

[6] Defined in Section 2.3.

Table 2. **UNI:** Approaches for Unifying Subtask Discovery with the Learning of Hierarchical Policy

Approach	How are subtasks discovered?	Definition of a subtask ω					No. of subtasks	Suitable task domains		Main utilities	Main limitations
		I_ω	π_ω	β_ω	g_ω	r_ω		state space	action space		
Unified Learning of Policy Tree											
Daniel et al. [17]	by learning a graphical model to segment trajectories into Options	global	learned using the task rewards	learned	n.r.	n.r.	PRE	D/C, S	D/C	learns trajectory-based Options from scratch	Options are not available during initial learning stage // number of Options needs to be predefined
Skill Chaining; Konidaris et al. [50]	by learning a classifier to identify initiation states of one Option ω and setting them as subgoals of another Option ω', i.e., $g_{\omega'} = I_\omega$	learned using the classifier	learned using r_ω	at g_ω	learned using the classifier	w.r.t. g_ω	DISC	D/C, S	D	learns a flexible number of subgoal-seeking Options from scratch	Options are not available during initial learning stage // works only for goal-directed tasks
Option-Critic; Bacon et al. [3]	learned using policy-gradients derived using the task rewards	global	learned using the task rewards	learned	n.r.	n.r.	PRE	D/C, L/S	D	learns general Options from scratch, starting from initial learning stage	number of Options needs to be predefined // performs poorly on sparse reward tasks [99]
Proximal Policy Option-Critic; Klissarov et al. [48]	learned using policy-gradients derived using the task rewards	global	learned using the task rewards	learned	n.r.	n.r.	PRE	D/C, L/S	C	extends Option-Critic to continuous action tasks	number of Options needs to be predefined // performs poorly on sparse reward tasks [4, 69]
Hierarchical Option-Critic; Riemer et al. [78]	learned using policy-gradients derived using the task rewards	global	learned using the task rewards	learned	n.r.	n.r.	PRE	D/C, L/S	D	extends Option-Critic to arbitrary number of levels of the hierarchical policy	number of Options needs to be predefined // might perform poorly on sparse reward tasks
Interest-Option-Critic; Khetarpal and Precup [46]	learned using policy-gradients derived using the task rewards	learned	learned using the task rewards	learned	n.r.	n.r.	PRE	D/C, L/S	D	adds Interest Function to Option-Critic for learning the initiation condition I_ω	number of Options needs to be predefined // might perform poorly on sparse reward tasks
Double Actor-Critic; Zhang et al. [104]	learned using policy-gradients derived using the task rewards	global	learned using the task rewards	learned	n.r.	n.r.	PRE	D/C, L/S	D/C	learns general Options using off-the-shelf policy optimization algorithms instead of algorithms customized for Option-based SMDP	number of Options needs to be predefined // might degenerate to frequently terminating Options due to the lack of deliberation cost

(Continued)

Table 2. Continued

Approach	How are subtasks discovered?	Definition of a subtask ω					No. of subtasks	Suitable task domains		Main utilities	Main limitations
		I_ω	π_ω	β_ω	g_ω	r_ω		state space	action space		
Unified Learning of Feudal Hierarchy											
FuN; Vezhnevets et al. [99]	by learning the subgoal space using policy gradients	global	learned using r_ω	after fixed no. of steps or at g_ω	learned	w.r.t. g_ω	DISC	D/C, L/S	D	learning low-dimensional subgoal space	learned representation of the subgoal space may not be optimal
Nachum et al. [70]	by learning the subgoal space using a sub-optimality bound	global	learned using r_ω	after fixed no. of steps or at g_ω	learned	w.r.t. g_ω	DISC	C, L/S	C	learning low-dimensional subgoal space, with theoretical guarantee of reduced sub-optimality	not expressive enough to represent subgoal-free behavior (unlike Options), e.g., drive through traffic, move in a circle, running, etc.

w.r.t. = *with respect to*; n.r. = *not required.*
D = discrete, C = continuous, S = small, L = large. PRE = predefined, DISC = discovered.
All the task domains are assumed to have moderate to extreme reward sparsity.

In all the approaches discussed above, an agent is not able to use the subtasks (Options) during the initial episodes of learning. This is because the agent must first collect experience trajectories to use for discovering the initial set of Options. Bacon et al. [3] propose an Option framework that can learn the Options along with the entire hierarchical policy from the beginning of the learning process. This approach is called **Option Critic (OC)**. In OC, a fixed number of Options are randomly initialized with parameterized policies and termination functions. The higher-level policy is also randomly initialized. Then, the higher-level policy and all the Options (policies and termination functions) are learned using policy gradients derived using the main task rewards. OC does not use subgoals or Option-specific rewards, and it is theoretically guaranteed to learn optimal hierarchical policy using the policy gradients. In evaluation, OC outperforms standard RL in Atari games such as Seaquest, Ms Pacman, and so on, using deep learning. It also performs better than skill chaining [50] in the initial stages of learning in the Pinball domain. A limitation of OC is that it requires the *number* of Options to be predefined. Moreover, the policy gradients are heavily dependent on the main task reward. Thus, OC is likely to perform poorly on the sparse-reward tasks [69, 99]. OC has become a popular framework for unified HRL due to its strong theoretical foundation. This has led to the emergence of various approaches that build upon OC [46, 48, 78], which are listed in Table 2.

The **Option-Critic (OC)** framework requires the policy optimization algorithms that are customized for the Option-based **Semi-Markov Decision Process (SMDP)** [93]. This puts a restriction on using other advanced policy optimization algorithms [34, 82] for HRL in an off-the-shelf manner. To address this issue, Zhang et al. [104] proposed the formulation of an HRL hierarchy as two parallel *augmented* **Markov Decision Processes (MDPs)** [55], where each MDP uses an augmented state space that is a cross-product of the original state space and the set of Options. The higher-level MDP models the problem of learning a policy over Options and their corresponding termination conditions, while the lower-level MDP models the problem of learning the Option policies. Both MDPs use the task rewards only, without subgoals or subtask rewards. The proposed approach is called **Double Actor Critic (DAC)**, because it applies actor-critic algorithms to learn

the policies at both the levels. Due to the augmented MDP formulation, DAC is compatible with various off-the-shelf actor-critic algorithms for policy optimization. It shows significantly better performance than OC on various robot simulation tasks. However, similar to OC, DAC requires the number of Options to be pre-defined.

It has been observed that OC and DAC approaches might encounter a higher-level policy *degeneration* phenomenon [36, 104] when the probability of termination of each Option becomes very high after a long period of training. This causes the higher-level policy to switch from one Option to another at almost every timestep, and the learned Options do not specialize on any recognizable behavior. Harb et al. [36] argue that temporally extended Options are not necessary from the theoretical perspective of policy optimization, because the optimal policy is actually achieved in terms of the primitive actions. To learn temporally extended Options, they introduced a regularizer called *deliberation cost*, which is a penalty received by the higher-level policy upon switching Options. This encourages the higher-level policy to retain each Option for a longer period of time, which leads to an *empirically* better performance than the frequently switching Options. The deliberation cost is included in the OC framework, but the authors of DAC leave the integration of such a cost for future work. Another form of degeneration might occur when the higher-level policy selects a single Option for the entire task duration. This is possibly because each Option can be initiated anywhere in the state space if it uses a global initiation condition I_ω. Khetarpal and Precup [46] proposed an approach called **Interest-Option-Critic (IOC)**, built on top of the OC framework, which learns the initiation condition of each Option using the policy gradients so various Options can specialize on different behaviors, thereby preventing the degeneration.

3.2.2 Unified Learning of Feudal Hierarchy. In a feudal UNI hierarchy, the subgoal space and the universal policies at all the levels are learned in a unified manner.

Vezhnevets et al. [99] propose a feudal hierarchy of neural networks in which a higher level network called the "Manager" samples a subgoal in a *learned* latent subgoal space. The subgoal may be a point in the latent space or it may be a unit vector representing a direction in the latent space. The subgoal is taken as input by a lower-level network called the "Worker," which must learn a policy to achieve the subgoal using the distance to the subgoal as a reward. The Worker is trained using the usual policy gradients derived from the subgoal-based rewards. The Manager is trained using a *transition gradient* introduced by the authors. This gradient is derived using the task reward as well as the distance between the subgoal assigned to the Worker and the actual state-transition made by the Worker. Thus, the Manager learns from both the task rewards and the Worker's behavior. The transition gradient is used to learn both the Manager policy and the latent subgoal space. This approach, called **Feudal Networks (FuN)**, shows better performance on Atari games, such as Montezuma's Revenge, than DQN [68] and Option-Critic [3].

Feudal Networks do not guarantee that the learned subgoal space leads to optimal hierarchical policy. Nachum et al. [70] address this issue in a subgoal representation learning approach developed upon HIRO [69] (HIRO is discussed in Section 3.1.1). The authors derive an optimization objective based on a theoretical bound on the sub-optimality of the hierarchical policy. This objective is used to learn a function $f_\theta(s)$ that transforms the state space to a low-dimensional subgoal space. Thus, the learned subgoal space representation minimizes the sub-optimality and the hierarchical policy based on HIRO addresses the non-stationarity issue. This approach is evaluated on MuJoCo continuous control tasks [97], using both low-dimensional state spaces and high-dimensional state spaces (e.g., images used as states). It outperforms various other subgoal representation schemes such as direct use of the original states (e.g., images) as subgoals, using latent space learned in the style of FuN [99], using subgoal embedding derived from Variational Autoencoder [47], and so on.

3.3 Independent Subtask Discovery (ISD)

This subsection reviews the approaches for independent subtask discovery for single agent and single task setting. The ISD challenge arises when the aim is to automatically find the subtasks that are task-agnostic and transferable to HRL agents across various unknown tasks. The implication is that the process of subtask discovery should be *independent* of the process of learning the hierarchical policy of an agent. The common idea behind the ISD approaches is similar to that of pre-training in the general context of machine learning [22]. In an ISD approach, subtask discovery typically occurs in a pre-training stage. The discovered subtasks are then used to learn a hierarchical policy to perform a particular target task. This pre-training approach leads to a few key issues that are part of ISD, as follows: ensuring that the subtask discovery process is data-efficient, discovering subtasks that allow diverse exploration of the state space independent of any specific task, learning continuous subtask space that allows generalization through sampling of new subtasks on target tasks, among other issues.

It is important to make a *disclaimer* that we do not strongly classify a particular approach as ISD without any possibility of being applicable for subtask discovery in a unified manner. A subtask discovery algorithm provided as part of an ISD approach may also be used for unified learning by running it simultaneously with the training of the agent's hierarchical policy, using the same task as the source of data for both. This is mentioned wherever applicable in the following discussion. Please refer to Table 3 for a summary of the ISD approaches.

3.3.1 Discovery of Subgoals. A set of subtasks can be constructed by discovering their corresponding subgoals (one of the objective components mentioned in Section 2.2.1) and then learning one subtask policy to reach one subgoal or a set of related subgoals.

A straightforward approach for subgoal discovery is to find the bottlenecks in the state space of a task domain. **Bottlenecks** are those states at which several paths connecting various initial states to various goal states converge. Removing the bottlenecks is likely to cut access to the possible goal states in the state space. Hence, the discovered bottlenecks are treated as important subgoals and policies of subtasks are learned to reach them.

McGovern and Barto [64] proposed a frequency-based approach for bottleneck discovery. This approach uses the idea of Diverse Density [63] to find interesting bottlenecks. An agent explores the state space using an initial policy and collects the trajectories from various initial states to various terminal states. Each trajectory is classified as *positive* or *negative,* depending on whether its terminal state is a desired goal state or not. Here, a desired goal state is related to a pre-training task. Then, a **Diverse Density (DD)** measure is used to estimate the probability of a state occurring on the positive trajectories vs. the probability of occurrence on the negative trajectories. The states with higher DD are chosen as the bottleneck subgoals. Simsek and Barto [84] also proposed a frequency-based bottleneck discovery approach that may be more data-efficient, because they build partial state-action transition graph from both the successful and unsuccessful experiences. In this graph, each node is a state and each edge is an action. The graph can be used to *simulate* new trajectories by simply following the connections between various nodes. The authors use a measure called *betweenness* defined as $\sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}} w_{st}$, where σ_{st} is the total number of optimal paths from a node s to a node t and $\sigma_{st}(v)$ is the number of such paths passing through a node v. w_{st} is the weight of a (s, t) pair. If the betweenness score for v is higher than its neighbors by a certain threshold, then v is chosen as a subgoal.

Frequency statistics require a large number of trajectories to be generated to estimate the occurrence frequencies of various states. A more efficient approach is to simply construct a state-action transition graph once, in which each node is a state and each edge is an action, and then find the bottlenecks by analyzing the connectivity between different sub-graphs. Menache et al. [66] proposed an approach called *Q-cut*, which builds a state-action transition graph over the global

Table 3. **ISD:** Approaches for Independent Subtask Discovery

Approach	How is a subtask discovered?	Suitable task domains		Main utilities	Main limitations
		state space	action space		
Discovery of Subgoals					
McGovern et al. [64] // Şimşek et al. [84]	using frequently occurring bottlenecks* as subgoals	D, S	D	simple to implement frequency-based heuristic	might require excessive exploration to collect multiple trajectories for estimating the bottleneck frequencies
Q-cut; Menache et al. [66]	using graph-cut bottlenecks as subgoals	D, S	D	data-efficient bottleneck discovery using graphs over the global state-action space	not easily scalable to continuous or large state spaces
L-cut; Şimşek et al. [85]	using graph-cut bottlenecks as subgoals	D, L/S	D	data-efficient bottleneck discovery using local sub-graphs	not easily scalable to continuous state spaces
Machado et al. [60]	using subgoals at local maxima of Proto Value Functions (PVFs) [61]	D/C, L/S	D	learning general subtasks for diverse exploration	computing PVFs may be computationally expensive
HASSLE; Bakker et al. [5]	using state space cluster centroids as subgoals	D, S	D	learning general subtasks for diverse exploration	not easily scalable to high-dimensional state spaces
HSP; Sukhbaatar et al. [88]	using a continuous latent embedding of subgoals learned via asymmetric self-play [89]	D/C, S/L	C	discovering continuous subgoal embedding instead of discrete subgoal space	cannot generalize to subgoal-free skills, e.g., drive through traffic, move in a circle, running, etc.
Discovery of Diverse Skills**					
VALOR; Achiam et al. [1]	by learning the policy π_ω to maximize the MI*** between the skill ω and the trajectories generated by executing it	D/C, S/L	D/C	discovers diverse skills and provides a procedure to grow the number of skills	requires excessive exploration data to maximize MI
Hausman et al. [38]	by learning the policy π_ω to minimize the cross-entropy between the skill embedding and the generated trajectories	D/C, S/L	C	discovers continuous embedding of skills instead of a discrete set	requires excessive exploration to learn both the skill embedding and the policy
SecTAR; Co-Reyes et al. [77]	by learning the policy π_ω to decode the latent embedding of the trajectories correlated with the skill ω	D/C, S/L	D	discovers continuous embedding of skills and additionally provides a technique for unified learning	requires excessive exploration to collect diverse trajectories for embedding

*defined in Section 3.3.1; **definition of "skill" is given in Section 2.3; ***Mutual Information.
w.r.t. = *with respect to*; n.r. = *not required*.
D = discrete, C = continuous, S = small, L = large. PRE = predefined.

environment and then finds bottleneck nodes using Max-Flow/Min-Cut algorithm [18]. Each bottleneck is treated as a subgoal for a subtask, and the subtask's policy is learned to reach it. Q-cut is not be easily scalable to large and/or continuous state spaces due to the need for constructing the *global* graph, which requires diverse exploration. Şimşek and Barto [85] proposed a similar graph-cut approach to find the bottlenecks, but they use local sub-graphs rather than a global graph constructed over the entire state space. This is called *L-cut*. L-cut can find local bottlenecks incrementally and can be applied on large state spaces, because an agent does not need to build the global graph. However, it can not be easily scaled to continuous state spaces, because the graphs used by the algorithm require discrete states as nodes.

Bottleneck discovery may not be suitable for task domains that do not necessarily contain identifiable bottlenecks. This may happen when the paths to various desired goal states are diversely

distributed and do not sufficiently converge on particular bottlenecks. Machado et al. [60] proposed an approach to find subgoals that are distributed over diverse regions of the state space rather than only at the bottlenecks. This approach uses **Proto Value Functions (PVFs)**[7] [61]. The subgoals lie at the local maxima of different PVFs defined over the state space. The PVFs are derived from the eigendecomposition of the Laplacian matrix[8] of the state transition graph. Bakker & Schmidhuber [5] proposed unsupervised clustering directly in the state space to discover subgoals over diverse regions. An agent explores the state space and forms clusters of states using an online **Adaptive Resource Allocation Vector Quantization (ARAVQ)** algorithm [57]. The centers of various clusters are used as subgoals. The exploration and clustering process is not dependent on any particular task, hence the subgoal discovery is task-agnostic or independent. The authors also provide an algorithm to learn a hierarchical policy using the already discovered subgoals. In this algorithm, subgoals constitute the action space of a higher-level policy that is learned using the task rewards. The lower level consists of multiple policies where each policy can specialize to reach a subset of the subgoal space and learn using the rewards for subgoal achievement. A mapping function is learned to assign a subgoal to the appropriate lower-level policy. There is also a provision for subgoal re-labelling, similar to HIRO [69] or HAC [54] (Section 3.1.1), for handling non-stationarity. The complete approach is called **Hierarchical Assignment of Subgoals to Sub-policies LEarning (HASSLE)**.

A significant limitation of all the subgoal discovery approaches reviewed so far is that the discovered subgoal space is *discrete*. In such a discrete space, it is not feasible to interpolate new subgoals, thereby limiting the range (or diversity) of the tasks that can be performed using the discovered subgoal space. **Discovery of continuous subgoal spaces** is important to address this issue. Sukhbaatar et al. [88] proposed an approach called **Hierarchical Self Play (HSP)** to learn continuous embedding of subgoals using asymmetric self-play [89]. Asymmetric self-play is an unsupervised pre-training phase that is illustrated as follows: At the beginning of asymmetric self-play, two standard (non-hierarchical) RL policies are initialized, say, Alice (π_A) and Bob (π_B). Let the initial state be s_0. First, Alice takes T_A steps in the environment starting from s_0. Let $s^* = s^A_{T_A}$ be the final state reached using Alice. Next, the environment is reset to s_0 and s^* is given as a target to Bob. At each timestep, s^* is encoded into a low-dimensional latent subgoal embedding using a learned encoder E, i.e., $g_t = E(s^B_t, s^*)$ where s^B_t is the state at time t while Bob is executing. The Bob policy selects primitive actions as $a^B_t = \pi_B(s^B_t, g_t)$. Bob is given T_B steps to achieve the target state s^*, otherwise it fails. It is given a reward $R^B = 1$ if the target is achieved, otherwise 0. Alice is given a reward $R_A = 1 - R_B$. The authors choose $T_B \sim T_A$. This results in an asymmetric self-play mechanism in which Bob learns to reach targets set by Alice and Alice learns to achieve new, unexplored targets that are beyond the reach of Bob. In this way, an agent consisting of these two policies explores the environment effectively without external supervision.

The subgoal encoder of HSP, $E(s^B_t, s^*)$, basically encodes various target states (s^*) into a low-dimensional subgoal space. Once a subgoal space has been discovered in the pre-training phase of asymmetric self-play, it is simply used as the *continuous* action space of a higher-level policy of an HRL agent used to perform a particular task. The lower level of the agent is initialized using the Bob policy and then fine-tuned on the task.

3.3.2 Discovery of Diverse Skills. The subgoal discovery approaches are unsuited to find subtasks without definite subgoals associated with them. Consider an example of a subtask "drive through traffic," which is a part of a longer horizon task of reaching a destination. This subtask

[7] PVFs are the basis functions of the state transition graph of an MDP.
[8] https://mathworld.wolfram.com/LaplacianMatrix.html.

requires an agent to maneuver a vehicle around traffic *without* any particular subgoal. Subgoal discovery is inapt for finding such a subtask. Therefore, general approaches are desired that can directly discover a set of diverse skills, instead of learning them through subgoals. As defined in Section 2.3, a "skill" refers to the policy of a subtask in the sense that it semantically represents the *ability to do something well*. After discovery, the skills can be added to an HRL agent as the lower-level policies and fine-tuned on a particular task.

One strategy for discovering diverse skills is by maximizing the **Mutual Information (MI)**[9] between a skill ω and the states reached or the trajectories generated by using that skill. This is achieved by using a latent skill vector, say, z_ω, and defining the skill policy as $\pi_\omega(s) = \pi(s, z_\omega)$, where $\pi(s, z_\omega)$ is a universal policy that takes the skill vector as input. z_ω is typically a one-hot encoded vector in most of such approaches [1, 24, 25, 31]. Maximizing the MI between z_ω and the states reached or trajectories generated by following $\pi(s, z_\omega)$ results in the discovery of a set of diverse skills correlated with diverse terminal states or trajectories. An overview of the specific approaches based on this strategy is provided below.

Florensa et al. [25] proposed an approach called SSN4HRL in which the universal policy $\pi(s, z_\omega)$ is represented using a stochastic neural network [98]. This policy is learned by maximizing an MI reward $P(z_\omega|(x, y))$, where (x, y) is the location of the agent in a spatial environment and $P(z_\omega|(x, y))$ is the probability of predicting z_ω by observing (x, y). This probability can only be maximized if the locations reached by executing different skills are sufficiently different. Gregor et al. [31] proposed **Variational Intrinsic Control (VIC)**, which discovers the skills by maximizing an objective containing an MI term $P(z_\omega|s_0, s_T)$ between the skill vector z_ω and the terminal state s_T reached by the policy $\pi(s, z_\omega)$, conditioned on the start state s_0. Eysenbach et al. [24] proposed an approach called **Diverity Is All You Need (DIAYN)**, which discovers the skills by optimizing an objective function designed to maximize the MI between a skill vector z_ω and every state in a trajectory generated by $\pi(s, z_\omega)$, without considering the order of the states in the trajectory.

All these approaches can discover a set of diverse skills but also share few common limitations, such as, they can only discover a predefined number of skills, and their MI objectives either ignore trajectories or do not consider the order of states in the trajectories. To address these limitations, Achiam et al. [1] proposed **Variational Autoencoding Learning of Options by Reinforcement (VALOR)**. In this approach, a policy $\pi(s, z_\omega)$ is basically an *encoder* that encodes z_ω into a trajectory. The trajectory is denoted as τ. It is fed into a *decoder* that must map it back to z_ω. This leads to an optimization objective aimed at maximizing $P(z_\omega|\tau)$. Thus, VALOR discovers a set of diverse skills that essentially correlate to a diversity of trajectories taken by an agent. VALOR also provides a procedure to discover a progressively increasing number of skills.

The ISD approaches discussed so far can only discover a *discrete* set of skills. A possible reason may be that computing the probability distributions over a continuous space of skills, for estimating the MI, is complicated. However, discovering a continuous space of skills is important for generalization, such that new skills can be easily interpolated in such a continuous space. Co-Reyes et al. [77] proposed an approach called **Self Consistent Trajectory Autoencoder (SeCTAR)**, in which an encoder LSTM [40] embeds the *state transition trajectories* (i.e., $\{s_t, s_{t+1}, s_{t+2} \ldots\}$) into a low-dimensional continuous latent vector space and a decoder LSTM learns to decode a latent vector into a policy. A latent vector represents similar trajectories, hence, the policy decoded from the latent vector is considered to represent a skill. A diverse latent space is learned by using an exploration mechanism that generates diverse trajectories. This encoder-decoder model is learned

[9]Mutual information (MI) of two random variables is the amount of information (in units such as *bits*) obtained about one variable by observing the other variable (https://people.cs.umass.edu/~elm/Teaching/Docs/mutInf.pdf).

in a pre-training phase for skill discovery. After the pre-training phase, the encoder module is removed and the encoded latent space is available as a continuous space of skills. A higher-level policy of an HRL agent samples a skill in this continuous space, which is then decoded by the decoder module in the form of a policy. In addition to independent skill discovery, the authors also extend SeCTAR for *unified learning* on a particular task. This is achieved via an *explorer policy* that explores the task relevant states and their neighborhood such that the trajectories encoded by the SeCTAR encoder module are correlated to the given task. This way, the discovered space of skills is situated in the task on which the HRL agent is being trained concurrently.

Hausman et al. [38] proposed a variational inference approach to learn diverse skills along with a continuous latent embedding of the skills. The objective used for learning is the minimization of the cross-entropy[10] between a distribution over the latent embedding conditioned on a one-hot skill vector and a distribution over the latent embedding conditioned on the trajectory generated by executing that skill. The learned latent embedding is the continuous space of skills in which new skills can be interpolated.

3.4 Transfer Learning with HRL (TransferHRL)

This subsection reviews the approaches for learning single HRL agent on multiple tasks via multi-task or transfer learning. The TransferHRL class represents all such approaches with or without subtask discovery. The concept of transfer learning with HRL arises from the idea that certain subtasks are shareable across multiple related tasks. The policy learned to perform a subtask in the context of one task can be used to accelerate the adaptation of the agent on other related tasks as well. Therefore, this is an important challenge to address in HRL.

Transfer learning is used in the context of RL by transferring experience data, action policy, or Q-value function from one task to another task to accelerate learning on the latter task [55, 81]. In the context of HRL, the problem of transferring the main task policy (the highest level in the hierarchical policy) can be addressed using the RL transfer approaches. However, the subtasks become the additional transferable components, including their policies, termination/initiation conditions, subgoals, and so on. This poses additional issues that are unique to the transfer learning with HRL (TransferHRL) compared to the transfer learning with standard RL. A few of the issues unique to TransferHRL are as follows:

- Efficiently scaling to a large number of subtasks during lifelong transfer learning.
- Transferring subtasks across task domains with different state spaces.
- Learning subtasks from scratch on multiple related tasks.

These issues are discussed henceforth in this subsection, along with the respective approaches to address them. We reiterate that the ISD approaches (Section 3.3) learn task-agnostic subtasks that can be transferred to various tasks. However, ISD approaches do not address the above-mentioned issues.

3.4.1 Transfer + Subtask Policy Distillation. An HRL agent might learn and accumulate a large number of subtask policies when transferring from one task to another, such as during lifelong learning [15]. Naively adding numerous subtasks to the action space of the higher-level policy may cause memory-inefficiency due to the need to store multiple subtask policies. Tessler et al. [96] proposed a deep HRL framework, called **Hierarchical Deep Reinforcement Learning Network (H-DRLN)**, for memory-efficient transfer and retention of a set of pre-trained policies corresponding to various subtasks (without subtask discovery). These policies are pre-trained using manually defined objectives. The policies are transferred to an HRL agent (in a target task) via

[10] Cross-entropy is a measure of the difference between two probability distributions for a given random variable.

multi-skill distillation introduced by the authors. Multi-skill distillation is a form of policy distillation [81] that enables H-DRLN to efficiently combine the policies of several subtasks into a single distilled policy. This makes H-DRLN memory-efficient. When performing a target task, H-DRLN agent uses a higher-level policy to choose a subtask that is executed by the distilled policy (at the lower level). H-DRLN learns to perform multiple tasks in the Minecraft game[11] with lower sample complexity and better performance compared to standard RL. The main limitation of H-DRLN is that the policies of subtasks need to be pre-trained using manually defined objectives (before distillation). Moreover, H-DRLN enables distillation of the existing policies into a target policy but does not provide a method for continual discovery of new subtasks.

3.4.2 Transfer + Agent-space Subtasks. A subtask policy learned in a particular task domain is conditioned on the state space S of that particular domain. If that policy is transferred to a new task domain whose state space is different from S, then the policy may no longer be optimal. To address this issue, it is necessary to achieve domain-invariant transfer such that the subtask policies are robust to the varying state spaces. Konidaris et al. [49] proposed an approach to learn Options that can be transferred to various tasks with different state spaces. An Option is a subtask representation as defined in Section 3.1.2. The invariance is achieved by using two separate state representations—one in the task-space and the other in the agent-space. The agent-space representation contains features that are dependent on the agent and invariant across various tasks. Option policies are learned in the agent-space and they can be transferred to various tasks with different task-spaces as long as the agent-space remains the same across those tasks. The higher-level policy, which selects an Option for execution, is learned using the task-space and adapts according to the active task. The authors design multiple versions of their own task domains to evaluate this method. It is found that introducing agent-space Options improves the agent's performance compared to regular Option-based techniques [93]. The main limitations of this approach are that the Options are not automatically discovered, since their policies are learned using pre-defined rewards and the distinction of the agent-space from task-space needs to be manually defined.

3.4.3 Transfer + Meta-learning of Subtasks. Apart from efficiently transferring pretrained subtasks, it is also important to discover subtasks across multiple tasks from scratch. Frans et al. [29] proposed a deep HRL approach for discovering Options via meta-learning on multiple related tasks. This approach, called **Meta Learning Shared Hierarchies (MLSH)**, contains a set of Options with parameters shared across different tasks, which are learned from scratch by jointly maximizing the expected rewards obtained in all the tasks (meta-learning). MLSH does not use any other handcrafted rewards or subgoals. The learned Options are transferred to the unseen tasks by using task-specific higher-level policies. MLSH is evaluated in the MuJoCo [97] locomotion domain on which it outperforms standard RL and Option-Critic [3]. The main limitation of MLSH is that the number of Options needs to be fixed, which limits the diversity of tasks that can be performed using MLSH. Simply increasing the number of Options would require an agent to store and learn a large number of Option policies, which is likely to be inefficient in terms of memory and computation.

3.5 Multi-agent Hierarchical Reinforcement Learning (MAHRL)

This subsection reviews the approaches for learning to coordinate multiple HRL agents on a single joint task. The MAHRL class represents all such approaches with or without subtask discovery. In the standard **Multi-Agent Reinforcement Learning (MARL)** [26, 27, 59, 94], multiple agents learn to coordinate their primitive action policies by optimizing a joint task

[11] https://github.com/vkurenkov/hierarchical-skill-acquisition.

objective and, occasionally, by sharing common information about their states and actions. Going beyond standard MARL, the benefits of task decomposition can also be exploited for multi-agent coordination by dividing a joint task into multiple subtasks distributed across different HRL agents. Each agent can perform more than one subtask and different agents learn to coordinate their higher-level policies (that choose the subtasks) rather than just the primitive action policies. In certain cases, the agents may only coordinate at the higher level while treating their primitive action policies as independent of each other [58]. Learning to coordinate among HRL agents is called **Multi-agent HRL (MAHRL)** [30].

Developing MAHRL agents is a non-trivial challenge due to different complex issues involved. All the usual issues found in the standard MARL also apply to MAHRL, such as the non-stationarity due to multiple agents learning simultaneously, partial observability due to limited communication, and difficulty of reward distribution in case of heterogeneous agents. However, the use of hierarchical policies (for all the agents) also introduces a few additional issues that are unique to MAHRL, described as follows:

- **Synchronization of subtask terminations across different agents.** The synchronization of the decisions made by different agents is affected by the terminations of their subtasks (i.e., β_ω) [79]. The subtasks of all the agents can be terminated in a *synchronized* manner by forcefully terminating all of them when the subtask of any one agent terminates or introducing a waiting time until all the subtasks terminate naturally. This scheme makes it easy to coordinate decisions but also puts a forced bias on the termination conditions, which may cause sub-optimal behavior. Another scheme is *asynchronous* termination in which each agent's subtask terminates independently of the terminations of other agents' subtasks. This scheme is free from any forced interruption but makes it difficult to coordinate decisions as they occur asynchronously.
- **Subtask discovery in the context of other agents' behaviors.** An agent must account for the presence of other agents in the environment during subtask discovery. Since other agents are non-stationary elements of an agent's environment, the discovered subtask space itself may become non-stationary. This also makes **independent subtask discovery (ISD)** quite challenging, because the applicability of the discovered subtasks of an agent depends not only on the target task dynamics but also on the other agents present in a target task.

Definitions of key terms and concepts. A few important terms and concepts used in this subsection, *in the context of HRL*, are defined below.

Centralized agents implies a paradigm in which all the agents share the highest level of their hierarchical policies using a single central policy (e.g., a central manager policy [2]).

Decentralized agents with centralized learning implies a paradigm in which all the agents have their own separate hierarchical policies (no central policy), but they learn by centralized information sharing (or communication). The information generally consists of the state observations and the subtasks being executed by different agents.

Fully decentralized agents implies a paradigm in which all the agents have decentralized hierarchical policies and learn *without* information sharing or central policies, under partial observability.[12]

Homogeneous agents are those agents that have similar subtasks spaces and hierarchical structures. This implies that each agent may have the capability to independently perform the task and there may not be significant inter-dependencies among different agents.

[12]Entire information about the task environment, including the activity of other agents, is not available to an agent.

Heterogeneous agents are those agents that have different subtask spaces and hierarchical structures. Hence, they are different in their capabilities and no individual agent may perform the entire task on its own. The agents may have strong inter-dependencies such that one agent assists another agent or it is impossible for an agent to receive task rewards unless another agent finishes its subtasks.
The key MAHRL approaches are discussed henceforth. Please refer to Table 4 for a summary of the approaches.

3.5.1 Learning Multi-agent Hierarchical Policies. Ghavamzadeh et al. [30] proposed an approach for cooperation among homogeneous agents at different levels of their task decomposition hierarchies. This approach is called *Cooperative HRL*. The authors use MAXQ to define and learn the hierarchical policy of each agent. This approach is developed upon an earlier work by Makar et al. [62]. The subtask space for each agent is predefined, using handcrafted reward function for each subtask. The task decomposition hierarchy of each agent may consist of more than two levels and each level can be defined as a cooperative or independent level. The agents cooperate in terms of their subtasks. The primitive action level is independent for all the agents. Moreover, the subtasks of different agents terminate using predefined *asynchronous* events. This approach uses *decentralized agents with centralized learning* paradigm. All the agents are homogeneous, and they learn their decentralized hierarchical policies by sharing information about their local states and chosen subtasks. The authors report better performance of Cooperative HRL against standard MARL in an **Automated Guided Vehicle (AGV)** task for warehouse management and a Trash collection task. To ease the requirement of continuous information sharing, the authors also proposed a variant of this approach with communicate and non-communicate decisions at the higher levels of every agent's hierarchy.

Cooperative HRL addresses the issues of coordination and communication among homogeneous agents that generally do not have strong inter-dependencies. However, this approach is not trivial to apply in heterogeneous setting in which various agents perform different parts of a task and typically have strong inter-dependencies, such that few agents can reach the rewarding states while others take a preceding role to enable such agents. This implies that the task rewards may be too sparse or delayed for certain agents, making the learning process difficult. Engineering agent-specific rewards can be daunting in complex domains. Another limitation of the Cooperative HRL is that it uses termination events that need to be manually specified for all the subtasks.

Pateria et al. [74] proposed an approach based on Options [93] to address both the above-mentioned issues. This approach is called **Inter Subtask Empowerment based Multi-agent Options (ISEMO).** It exploits the inter-dependencies among heterogeneous agents to derive internal rewards called **Inter Subtask Empowerment Rewards (ISER)**. The inter-dependencies are based on the predefined initiation conditions of various subtasks across different agents. ISER is given as a positive reward to an agent that *empowers* another agent by creating the environment conditions that satisfy the initiation conditions of the latter agent's subtask(s). ISER is added to the (external) task reward observed by the agent, thereby augmenting the task reward. Hence, even if the task reward is sparse for a particular agent, it can still learn useful behavior using ISER for enabling other agents. Moreover, ISEMO uses the gradient-based learning of the termination function of each subtask on the lines of the Option-Critic [3]. The learned terminations are *asynchronous*. The authors compare ISEMO against the Cooperative HRL approach on a Search & Rescue task with a mix of heterogeneous and homogeneous agents. ISEMO is found to perform better than Cooperative HRL due to both ISER and the learned terminations. A similarity between ISEMO and Cooperative HRL is that ISEMO also uses the *decentralized agents with centralized learning* paradigm in which the agents share the information about their local states and chosen subtasks. The

Table 4. **MAHRL:** Approaches for Multi-Agent Hierarchical Reinforcement Learning

Approach	Paradigm	How is a subtask discovered?	Definition of a subtask ω					Suitable task domains			Main utilities	Main limitations
			I_ω	π_ω	β_ω	g_ω	r_ω	state space	action space	type of agents		
Learning Multi-Agent Hierarchical Policies												
Cooperative HRL; Ghavamzadeh et al. [30]	decentralized agents with centralized learning	-	PRE	learned using r_ω	PRE, *async*	n.r.	PRE	D, S	D	HOM	MAHRL for homogeneous agents, with flexible communication	restricted to homogeneous agents
ISEMO; Pateria et al. [74]	decentralized agents with centralized learning	-	PRE	PRE	learned, *async*	n.r.	n.r.	D/C, S/L	D	HOM/ HETR	MAHRL for heterogeneous or homogeneous agents, using a reward augmentation strategy based on inter-dependencies among the agents	the initiation conditions of various subtasks must be predefined to capture the inter-dependencies among the agents
Tang et al. [95]	decentralized agents with centralized learning	-	global	learned using r_ω	PRE, *sync or async*	n.r.	PRE	D/C, S/L	D	HOM	MAHRL for homogeneous agents using deep learning	restricted to homogeneous agents
FMH; Ahilan et al. [2]	centralized agents	-	global	learned using r_ω	PRE, *sync*	PRE	w.r.t. g_ω	D/C, S/L	D	HOM/ HETR	MAHRL using centralized control	learning the central high-level policy is likely to be complex if the joint subtask space is large
PoEM; Liu et al. [58]	fully decentralized agents	-	PRE	learned from pre-existing demon-strations	PRE, *async*	n.r.	n.r.	D, S	D	HOM/ HETR	MAHRL with full decentralization and under partially observability	requires preexisting demonstration trajectories
Learning Multi-Agent Hierarchical Policies in Unification with Subtask Discovery												
HSD; Yang et al. [101]	decentralized agents with centralized learning	by maximizing Mutual Information (MI) between subtasks and trajectories	global	learned using r_ω	PRE, *sync*	n.r.	MI + task re-ward	D/C, S/L	D	HOM	discovery of diverse skills unified with the learning of hierarchical policies of all agents	the number of skills needs to be predefined and fixed
DOC; Chakravorty et al. [14]	centralized agents	using policy gradients derived using the task reward	global	learned using task rewards	learned, *async*	n.r.	n.r.	D/C, S/L	D	HOM/ HETR	learning Options hierarchy in a unified manner for all agents	the number of Options needs to be predefined and fixed // may perform poorly if the task reward is sparse

w.r.t. = *with respect to*; n.r. = *not required*; *sync* = synchronous, *async* = asynchronous.
D = discrete, C = continuous, S = small, L = large. PRE = predefined, DISC = discovered, HOM = homogeneous, HETR = heterogeneous.
All the task domains are assumed to have moderate to extreme reward sparsity.

main limitation of ISEMO is that the initiation conditions of various subtasks need to be manually defined for generating ISER. Also, ISEMO only learns the termination functions of the subtasks but requires the policies of the subtasks to be predefined.

Tang et al. [95] proposed a *deep learning* approach for MAHRL with predefined subtasks (defined using handcrafted subtask rewards). They use *decentralized agents with centralized learning* by adapting deep MARL approaches such as QMIX [76] and CommNet [90] for MAHRL. Different schemes for *synchronous and asynchronous* terminations of different subtasks are also proposed. This approach is evaluated on a modified version of the Trash collection task [30] and an online mobile game. It is found to outperform standard MARL and independently learned HRL agents. However, it is only shown to work for homogeneous agents.

The approaches discussed above are applicable to decentralized agents with centralized learning. Another approach is to learn *centralized agents* that have a common central policy at the highest level. Ahilan et al. [2] proposed a deep MAHRL approach in which a central manager policy chooses subtasks for a set of workers, simultaneously. The subtasks are predefined using subgoals. It is a feudal hierarchy in which each worker policy is a universal policy. Each worker policy belongs to a separate agent. Only the central manager receives the task rewards, while the workers learn using the subgoal achievement rewards. This approach is called **Feudal Multi-agent Hierarchies (FMH)**. The workers can be heterogeneous or homogeneous. The terminations of different subtasks being performed by different workers are *synchronized* by the manager. FMH is evaluated on cooperative communication and navigation tasks in Multi-agent Particle Environments[13] and found to outperform standard MARL.

The approaches discussed so far assume that the information can be shared among different agents either via communication or by using central policy. A more challenging but realistic paradigm is learning to coordinate *fully decentralized* agents. This paradigm does not assume the availability of communication or centralization during training or execution. Hence, the agents need to learn under *partial observability* [44]. Omidshafiei et al. [72] introduced a theoretical framework for decentralized multi-agent coordination using subtasks (macro-actions) called **Decentralized Partially Observable Semi-Markov MDPs (Dec-POSMDPs)**. Liu et al. [58] provide a MAHRL approach based on Dec-POSMDPs. In this approach, the subtasks for all the agents are predefined and it is assumed that the agents have obtained a set of high-level trajectories from expert demonstrations containing sequences of subtasks. Then, the policy hierarchies of the decentralized agents are learned using an algorithm called **Policy-based Expectation Maximization (PoEM)** developed by the authors. All the policies take *history of state transitions* as input to resolve partial observability. PoEM is evaluated on multi-agent box pushing task and found to outperform dynamic-programming approaches. It works for *homogeneous or heterogeneous* agents. The subtask terminations can be *asynchronous*. However, a key limitation of this approach is that it requires the demonstration trajectories to be given in advance.

3.5.2 Learning Multi-agent Hierarchical Policies in Unification with Subtask Discovery. Yang et al. [101] proposed a MAHRL approach with skill[14] discovery based on variational inference [1] (Section 3.3.2). This approach is called **Hierarchical Skill Discovery (HSD)**. The paradigm is *decentralized agents with centralized learning* where the agents are *homogeneous*. HSD uses QMIX [76] for centralized learning. In HSD, each agent uses a decentralized feudal hierarchy consisting of two levels. It selects a one-hot encoded skill vector using its higher-level policy. The lower-level universal policy takes this skill vector as an input and generates trajectories. This universal policy

[13] https://github.com/openai/multiagent-particle-envs.
[14] Defined in Section 2.3.

is learned by maximizing the **Mutual Information (MI)** between various input skill vectors and the corresponding trajectories, resulting in the discovery of *diverse skills*. However, an agent must also align the skill discovery with the coordination with other agents. Hence, HSD adds the task reward to the MI objective of the lower-level universal policy so the learned skills are not only diverse but also grounded in the multi-agent task objective. The higher-level policy of each agent is learned using the task reward only. The skills of different agents terminate in a *synchronized* manner after fixed timesteps. HSD is evaluated on a soccer game in STS2 simulator.[15] It performs better than standard MARL and independently trained HRL agents. The authors also report few identifiable discovered skills such as moving for offense, moving for defense, taking shots, and so on. They also find that the skills discovered only with MI objective and without adding the task reward do not lead to good performance. This shows that the discovered skills *cannot be task-agnostic*. The main limitation of HSD is that the number of skills/subtasks has to be predefined and fixed. There is no technique provided to grow this number or learn a large space of skills.

Chakravorty et al. [14] proposed a MAHRL approach that extends the single-agent Option-Critic framework [3] for multi-agent coordination. This approach is called **Distributed Option-Critic (DOC**). They use a single central higher-level policy for all the agents (hence, *centralized agents* paradigm). The lower-level Option policies are decentralized. The agents can be heterogeneous or homogeneous. The agents also have a learned broadcast function to share the local information with the central higher-level policy. The higher-level policy, lower-level Option policies, and the Option termination functions are learned using policy gradients derived using the task reward *only*. The learned terminations are *asynchronous*. The authors evaluate DOC on simple grid-world tasks and find that it performs competitively against standard MARL.

3.6 Key Takeaways from the Survey of Approaches

Through the survey of the wide-ranging selection of HRL approaches in this section, we aim to provide important insights about the expansive progress and the current state of the HRL research. In this regard, a few key takeaways from the surveyed approaches are discussed henceforth.

There is no consolidated framework to learn hierarchical policies—different approaches should be used for different benefits. The two sub-classes of the approaches for learning hierarchical policies provide different but important benefits. The feudal hierarchy approaches use subgoals (or instructions) to represent various possible subtasks [19, 41, 51, 69]. Such approaches have the scope to scale up the subtask space by either using a large number of subgoals or using a low-dimensional *continuous* subgoal space. The scale-up is also possible because of the use of a universal policy at each level, which means that a large number of subtasks do not require many separate policies to be learned and stored, unlike the policy trees.

However, the policy tree approaches [73, 93] are not constrained to learn only subgoal-based (or instruction-based [41]) subtasks, unlike the feudal approaches. Different subtask policies can represent different types of behaviors such as subgoal-based, trajectory-based (e.g., "move in a circle"), general skills (e.g., "drive through traffic"), and so on. *To summarize, different approaches provide different benefits, but there is no consolidated framework that covers all the aspects.*

A variety of principles may be used for subtask discovery. There is no singular criterion for the quality of discovered subtasks. HRL researchers have a variety of ideas about the principle used for subtask discovery—partitioning state-action transition graphs [66, 85], clustering [5], diversifying exploration [60, 88], variational inference [1], and so on. These principles are mostly based on the objectives, which are different from the task reward maximization objective. Hence, the optimality of the discovered subtasks can not be theoretically guaranteed. The task

[15] https://github.com/electronicarts/SimpleTeamSportsSimulator.

reward may itself be used for subtask discovery, as shown by a few of the UNI approaches [3, 70, 99]. But, even among these approaches, there are different ways to include the reward into the subtask discovery process, e.g., using policy gradients [3, 99], using sub-optimality objective [70], and so on. Hence, *there is no standard principle for subtask discovery yet. The reader must gauge a particular approach based on the preferred characteristics.*

New trends may emerge along the direction of large-scale skill discovery. In the recent years, the reinforcement learning community has shown a growing interest in the continuous control and robotic manipulation problems [32, 35]. Following this interest, HRL research has also begun to produce approaches for the discovery of robotic skills [1, 77] and learning hierarchical policies for continuous control [54, 69, 102]. This trend might continue in the near future, especially concerning the large-scale acquisition of robotic continuous-control skills, such as "jumping," "running," "moving objects," "driving through traffic," "parking," and so on, in a diversity of task domains [83] and the adaptation of the UNI approaches for the end-to-end learning of a large collection of such skills. This trend is important, because diverse robotic skills are necessary for various real-life applications of HRL such as in warehouse management, surgical robots, autonomous driving, and so on.

4 OPEN PROBLEMS FOR FUTURE RESEARCH

HRL is garnering more interest in the reinforcement learning community. However, there still are different open problems concerning the scalability, efficiency, and theoretical robustness of HRL. A few such open problems are discussed henceforth, with the aim of identifying broader directions for future research. This discussion excludes the incremental issues and the extensions of any specific approach.

Building a lifelong knowledge base of transferable skills. Humans learn diverse skills during their lifetimes that are retained as valuable know-hows and selectively applied in different scenarios. The same faculty is essential for realizing the *general* HRL agents that can continuously learn, retain, and reuse a broad range of skills over their lifetimes. Building a lifelong knowledge base of skills requires the integration of multiple operations, such as skill discovery, skill transfer, and hierarchical learning using the relevant skills as subtasks to perform a new task. Currently, the necessary operations are provided by disparate approaches, specifically for unsupervised skill discovery [1, 38, 83], skill transfer [29, 96], skill composition [6], skill reuse [37], and selective initiation of skills while training the HRL agent [45]. *Further research and development is necessary to design an integrated HRL framework that holistically combines these operations for maintaining the lifelong knowledge bases of transferable skills.*

Leveraging high-level planning for improved learning and adaptation. Replacing the model-free higher-level policy of an HRL agent with model-based planning, using the learned subtask-to-subtask (or subgoal-to-subgoal) transition dynamics, can potentially accelerate the learning of the agent on the long-horizons tasks and the adaptation to new tasks with shared dynamics [23, 86]. Zahavy et al. [103] propose an approach to integrate high-level planning with HRL by using a set of sub-rewards to decompose the task reward. It is assumed that the sub-rewards are predefined. One subtask policy is defined for collecting one sub-reward. The higher-level planning heuristic finds the optimal sequence of subtasks by solving a Travelling Salesman Problem [11]. Sohn et al. [86] propose **neural subtask graph solver (NSGS)** for planning. This approach uses a subtask graph in which the subtasks, their preconditions, and their inter-dependencies are predefined. The higher-level planner is a graph solver, which plans over the subtask graph. The graph can be used to generalize to new tasks in a zero-shot fashion. The common limitation of both the approaches outlined above is that the subtasks and/or their inter-dependencies need to be

handcrafted. Eysenbach et al. [23] propose an approach called **Search on the Replay Buffer (SoRB)** that automatically extracts the subgoal states from the experience replay buffer[16] and builds a graph over them. In this graph, each node is a state and the weight of a transition edge between a pair of nodes is equal to the distance between the nodes estimated using the learned Q-value function. The higher-level planning heuristic finds the shortest-path over the subgoals in the graph to reach a goal. The path is traversed using the lower-level primitive-action policy trained via reinforcement learning. While SoRB shows performance improvements over model-free reinforcement learning, it requires the primitive-action policy to be trained before the higher-level graph can be constructed for planning. Hence, SoRB is not end-to-end unified, and it only addresses the problem of fast adaptation on the new goal-directed tasks after learning the primitive-action policy and the higher-level graph on the previous goal-directed tasks.

To summarize, there are various aspects to consider when integrating high-level planning with HRL. Automatic discovery of subtasks is paramount. *More investigation is required on learning the subtask inter-dependencies automatically, instead of handcrafting them. New approaches should also be developed for the end-to-end unification of higher-level planning and lower-level policy learning.*

Providing theoretical support for HRL. HRL delivers apparent practical (or empirical) benefits over standard RL in many long-horizon task domains [3, 51, 54, 69, 99]. In theory, however, there is no incentive for using a hierarchical policy in terms of Q-value maximization alone. Standard reinforcement learning, especially Q-learning [100], is theoretically guaranteed to converge to the desired maximum Q-value [65] using only a primitive action policy. Majority of the *empirical* performance benefits of HRL may simply be due to improved exploration [43, 71]. Nachum et al. [71], however, also show that a sophisticated non-hierarchical exploration scheme paired with a standard RL agent may provide equivalent performance as an HRL agent. Therefore, *more theoretical research needs to be done to support the empirical benefits of HRL* and *to identify the problem domains in which HRL provides clear advantage in terms of optimal performance.*

5 CONCLUSION

This survey provides a panoramic overview of the research done so far in the field of Hierarchical Reinforcement Learning (HRL), from the classical approaches to the recent advances. A novel and general taxonomy is devised that organizes the approaches into five broad classes based on the key challenges they address, which are: (i) Learning Hierarchical Policy (LHP), (ii) Learning Hierarchical Policy in Unification with Subtask Discovery (UNI), (iii) Independent Subtask Discovery (ISD), (iv) Transfer Learning with HRL (TransferHRL), and (v) Multi-agent HRL (MAHRL). Further classification of the approaches is also provided on the basis of the specific methodologies. Through the survey, we find that significant advances have been made with respect to addressing the two main sub-problems of HRL—how to learn a hierarchical policy and how to automatically discover subtasks—as mentioned in the HRL problem statement (Section 2.2.3). The field is now making strides in the directions of transfer learning and multi-agent learning using HRL, in which new trends may emerge, specifically concerning subtask discovery.

Despite this noteworthy progress, there is scope for further growth in HRL to encourage its wider acceptability as a scalable and robust paradigm. In this regard, we identify a set of important open problems to motivate the future research and advancements. These chosen problems concern: (i) lifelong skill discovery and utilization for scalable HRL, (ii) improving the data efficiency by leveraging high-level planning, and (iii) providing more theoretical guarantees of optimality of the HRL approaches.

[16] A storage of several experience data samples, of the form (s_t, a_t, r_t, s_{t+1}), used to train an agent.

REFERENCES

[1] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. 2018. Variational option discovery algorithms. arxiv:1807.10299 (2018).

[2] Sanjeevan Ahilan and Peter Dayan. 2019. Feudal multi-agent hierarchies for cooperative reinforcement learning. arxiv:1901.08492 (2019).

[3] Pierre-Luc Bacon, Jean Harb, and Doina Precup. 2017. The option-critic architecture. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*. AAAI Press, 1726–1734.

[4] Akhil Bagaria and George Konidaris. 2020. Option discovery using deep skill chaining. In *Proceedings of the 8th International Conference on Learning Representations*.

[5] Bram Bakker and Jürgen Schmidhuber. 2004. Hierarchical reinforcement learning with subpolicies specializing for learned subgoals. In *Proceedings of the IASTED International Conference on Neural Networks and Computational Intelligence*. IASTED/ACTA Press, 125–130.

[6] Andre Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Jonathan Hunt, Shibl Mourad, David Silver, and Doina Precup. 2019. The option keyboard: Combining skills in reinforcement learning. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc., 13052–13062.

[7] Andrew G. Barto and Sridhar Mahadevan. 2003. Recent advances in hierarchical reinforcement learning. *Discr. Event Dyn. Syst.* 13, 1-2 (2003), 41–77. DOI: https://doi.org/10.1023/A:1025696116075

[8] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. 1970. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Ann. Math. Statist.* 41, 1 (02 1970), 164–171. DOI: https://doi.org/10.1214/aoms/1177697196

[9] Melike Baykal-Gürsoy. 2010. Semi-Markov decision processes. *Wiley Encyclopedia of Operations Research and Management Science* (2010). DOI: https://doi.org/10.1002/9780470400531.eorms0757

[10] Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos. 2016. Unifying count-based exploration and intrinsic motivation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*. Curran Associates Inc., Red Hook, NY, 1479–1487.

[11] Richard Bellman. 1962. Dynamic programming treatment of the travelling salesman problem. *J. ACM* 9, 1 (Jan. 1962), 61–63. DOI: https://doi.org/10.1145/321105.321111

[12] Richard Bellman. 1954. The theory of dynamic programming. *Bull. Amer. Math. Soc.* 60, 6 (11 1954), 503–515.

[13] Mordechai Ben-Ari and Francesco Mondada. 2018. Finite state machines. In *Elements of Robotics*. Springer, 55–61. DOI: https://doi.org/10.1007/978-3-319-62533-1_4

[14] Jhelum Chakravorty, Patrick Nadeem Ward, Julien Roy, Maxime Chevalier-Boisvert, Sumana Basu, Andrei Lupu, and Doina Precup. 2020. Option-critic in cooperative multi-agent systems. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS'20)*. International Foundation for Autonomous Agents and Multiagent Systems, 1792–1794.

[15] Z. Chen and B. Liu. 2018. *Lifelong Machine Learning*. Vol. 12. Morgan & Claypool Publishers. 1–207 pages. DOI: https://doi.org/10.2200/S00737ED1V01Y201610AIM033

[16] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of the 8th Workshop on Syntax, Semantics and Structure in Statistical Translation*. Association for Computational Linguistics, 103–111. DOI: https://doi.org/10.3115/v1/W14-4012

[17] Christian Daniel, Herke Van Hoof, Jan Peters, and Gerhard Neumann. 2016. Probabilistic inference for determining options in reinforcement learning. *Mach. Learn.* 104, 2-3 (2016), 337–357. DOI: https://doi.org/10.1007/s10994-016-5580-x

[18] G. Dantzig and Delbert Ray Fulkerson. 2003. On the max flow min cut theorem of networks. *Lin. Ineq. Relat. Syst.* 38 (2003), 225–231.

[19] Peter Dayan and Geoffrey E. Hinton. 1993. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*, Vol. 5. Morgan-Kaufmann, 271–278.

[20] Thomas G. Dietterich. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Int. Res.* 13, 1 (Nov. 2000), 227–303.

[21] Mostafa Al-Emran. 2015. Hierarchical reinforcement learning: A survey. *Int. J. Comput. Dig. Syst.* 4, 02 (2015). DOI: https://doi.org/10.12785/IJCDS/040207

[22] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. 2010. Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.* 11 (Mar. 2010), 625–660.

[23] Ben Eysenbach, Russ R. Salakhutdinov, and Sergey Levine. 2019. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc., 15246–15257.

[24] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. 2018. Diversity is all you need: Learning skills without a reward function. arxiv:1802.06070 (2018).
[25] Carlos Florensa, Yan Duan, and Pieter Abbeel. 2017. Stochastic neural networks for hierarchical reinforcement learning. arxiv:1704.03012 (2017).
[26] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. 2016. Learning to communicate with deep multi-agent reinforcement learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*. Curran Associates Inc., Red Hook, NY, 2145–2153.
[27] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2017. Counterfactual multi-agent policy gradients. arxiv:1705.08926 (2017).
[28] Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. 2017. Multi-level discovery of deep options. arxiv:1703.08294 (2017).
[29] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. 2017. Meta learning shared hierarchies. arxiv:1710.09767 (2017).
[30] Mohammad Ghavamzadeh, Sridhar Mahadevan, and Rajbala Makar. 2006. Hierarchical multi-agent reinforcement learning. *Auton. Agents Multi-agent Syst.* 13, 2 (2006), 197–229. DOI: https://doi.org/10.1007/s10458-006-7035-4
[31] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. 2016. Variational intrinsic control. arxiv:1611.07507 (2016).
[32] Shixiang Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. 2016. Deep reinforcement learning for robotic manipulation. *CoRR* abs/1610.00633 (2016).
[33] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. 2020. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Proceedings of the Conference on Robot Learning (Proceedings of Machine Learning Research)*, Vol. 100. PMLR, 1025–1037.
[34] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 80. PMLR,, 1861–1870.
[35] Tuomas Haarnoja, Aurick Zhou, Sehoon Ha, Jie Tan, George Tucker, and Sergey Levine. 2018. Learning to walk via deep reinforcement learning. (2018). arxiv:1812.11103
[36] Jean Harb, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup. 2018. When waiting is not an option: Learning options with a deliberation cost. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.
[37] Leonard Hasenclever, Fabio Pardo, Raia Hadsell, Nicolas Heess, and Josh Merel. 2020. CoMic: Complementary task learning & mimicry for reusable skills. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 119. PMLR, 4105–4115.
[38] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin A. Riedmiller. 2018. Learning an embedding space for transferable robot skills. In *Proceedings of the 6th International Conference on Learning Representations*.
[39] Bernhard Hengst. 2010. Hierarchical reinforcement learning. In *Encyclopedia of Machine Learning*. Springer US, Boston, MA, 495–502. DOI: https://doi.org/10.1007/978-0-387-30164-8_363
[40] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780. DOI: https://doi.org/10.1162/neco.1997.9.8.1735
[41] YiDing Jiang, Shixiang (Shane) Gu, Kevin P. Murphy, and Chelsea Finn. 2019. Language as an abstraction for hierarchical deep reinforcement learning. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc., 9419–9431.
[42] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. 2016. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. arxiv:1612.06890 (2016).
[43] Nicholas K. Jong, Todd Hester, and Peter Stone. 2008. The utility of temporal abstraction in reinforcement learning. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)*. International Foundation for Autonomous Agents and Multiagent Systems, 299–306.
[44] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artif. Intell.* 101, 1–2 (May 1998), 99–134.
[45] Khimya Khetarpal, Martin Klissarov, Maxime Chevalier-Boisvert, Pierre-Luc Bacon, and Doina Precup. 2020. Options of interest: Temporal abstraction with interest functions. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*. AAAI Press, 4444–4451.
[46] Khimya Khetarpal and Doina Precup. 2019. Learning options with interest functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 9955–9956. DOI: https://doi.org/10.1609/aaai.v33i01.33019955
[47] Diederik P. Kingma and Max Welling. 2014. Auto-encoding variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations*.

[48] Martin Klissarov, Pierre-Luc Bacon, Jean Harb, and Doina Precup. 2017. Learnings options end-to-end for continuous action tasks. *CoRR* abs/1712.00004 (2017).

[49] George Konidaris and Andrew Barto. 2007. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI'07)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 895–900.

[50] George Konidaris and Andrew Barto. 2009. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems (NIPS'09)*. Curran Associates Inc., Red Hook, NY, 1015–1023.

[51] Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*. Curran Associates Inc., Red Hook, NY, 3682–3690.

[52] Tejas D. Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J. Gershman. 2016. Deep successor reinforcement learning. arxiv:1606.02396 (2016).

[53] Alessandro Lazaric. 2012. Transfer in reinforcement learning: A framework and a survey. In *Reinforcement Learning*. Springer, 143–173. DOI: https://doi.org/10.1007/978-3-642-27645-3_5

[54] Andrew Levy, George Dimitri Konidaris, Robert Platt Jr., and Kate Saenko. 2019. Learning multi-level hierarchies with hindsight. In *Proceedings of the 7th International Conference on Learning Representations*.

[55] Kfir Y. Levy and Nahum Shimkin. 2011. Unified inter and intra options learning using policy gradient methods. In *Proceedings of the 9th European Conference on Recent Advances in Reinforcement Learning (EWRL'11)*. Springer-Verlag, Berlin, 153–164. DOI: https://doi.org/10.1007/978-3-642-29946-9_17

[56] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *Proceedings of the 4th International Conference on Learning Representations*.

[57] Fredrik Linåker. 2000. Time series segmentation using an adaptive resource allocating vector quantization network based on change detection. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*. IEEE Computer Society, 6323. DOI: https://doi.org/10.1109/IJCNN.2000.859416

[58] Miao Liu, Christopher Amato, Emily P. Anesta, J. Daniel Griffith, and Jonathan P. How. 2016. Learning for decentralized control of multiagent systems in large, partially-observable stochastic environments. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI'16)*. AAAI Press, 2523–2529. Retrieved from https://dl.acm.org/doi/10.5555/3016100.3016253.

[59] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. Curran Associates Inc., Red Hook, NY, 6382–6393.

[60] Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. 2017. A Laplacian framework for option discovery in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*. JMLR.org, 2295–2304.

[61] Sridhar Mahadevan and Mauro Maggioni. 2007. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *J. Mach. Learn. Res.* 8 (Dec. 2007), 2169–2231.

[62] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. 2001. Hierarchical multi-agent reinforcement learning. In *Proceedings of the 5th International Conference on Autonomous Agents (AGENTS'01)*. Association for Computing Machinery, New York, NY, 246–253. DOI: https://doi.org/10.1145/375735.376302

[63] Oded Maron and Tomás Lozano-Pérez. 1998. A framework for multiple-instance learning. In *Advances in Neural Information Processing Systems*, Vol. 10. The MIT Press, 570–576.

[64] Amy McGovern and Andrew G. Barto. 2001. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the 18th International Conference on Machine Learning (ICML'01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 361–368.

[65] Francisco S. Melo. 2001. *Convergence of q-learning: A Simple Proof*. Technical Report. Institute of Systems and Robotics.

[66] Ishai Menache, Shie Mannor, and Nahum Shimkin. 2002. Q-cut—Dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the 13th European Conference on Machine Learning (ECML'02)*. Springer-Verlag, Berlin, 295–306.

[67] Matheus R. F. Mendonça, Artur Ziviani, and André M. S. Barreto. 2019. Graph-based skill acquisition for reinforcement learning. *ACM Comput. Surv.* 52, 1 (Feb. 2019). DOI: https://doi.org/10.1145/3291045

[68] 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. DOI: https://doi.org/10.1038/nature14236

[69] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. 2018. Data-efficient hierarchical reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates Inc., Red Hook, NY, 3307–3317.

[70] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. 2018. Near-optimal representation learning for hierarchical reinforcement learning. arxiv:1810.01257 (2018).

[71] Ofir Nachum, Haoran Tang, Xingyu Lu, Shixiang Gu, Honglak Lee, and Sergey Levine. 2019. Why does hierarchy (sometimes) work so well in reinforcement learning? arxiv:1909.10618 (2019).

[72] S. Omidshafiei, A. Agha-mohammadi, C. Amato, and J. P. How. 2015. Decentralized control of partially observable Markov decision processes using belief space macro-actions. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'15)*. 5962–5969. DOI: https://doi.org/10.1109/ICRA.2015.7140035

[73] Ronald Parr and Stuart Russell. 1998. Reinforcement learning with hierarchies of machines. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS'97)*. The MIT Press, Cambridge, MA, 1043–1049.

[74] Shubham Pateria, Budhitama Subagdja, and Ah-Hwee Tan. 2019. Multi-agent reinforcement learning in spatial domain tasks using inter subtask empowerment rewards. In *Proceedings of the IEEE Symposium Series on Computational Intelligence*. IEEE, 86–93. DOI: https://doi.org/10.1109/SSCI44817.2019.9002777

[75] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*. JMLR.org, 2778–2787.

[76] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. 2018. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. arxiv:1803.11485 (2018).

[77] John D. Co-Reyes, Yuxuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel, and Sergey Levine. 2018. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. arxiv:1806.02813 (2018).

[78] Matthew Riemer, Miao Liu, and Gerald Tesauro. 2018. Learning abstract options. In *Advances in Neural Information Processing Systems*, Vol. 31. Curran Associates, Inc., 10424–10434.

[79] Khashayar Rohanimanesh and Sridhar Mahadevan. 2002. Learning to take concurrent actions. In *Proceedings of the 15th International Conference on Neural Information Processing Systems (NIPS'02)*. The MIT Press, Cambridge, MA, 1651–1658.

[80] Stuart Russell and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall Press.

[81] Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. 2016. Policy distillation. In *Proceedings of the 4th International Conference on Learning Representations*.

[82] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 37. PMLR„ 1889–1897.

[83] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. 2019. Dynamics-aware unsupervised discovery of skills. arxiv:1907.01657 (2019).

[84] Özgür Şimşek and Andrew G. Barto. 2008. Skill characterization based on betweenness. In *Proceedings of the 21st International Conference on Neural Information Processing Systems (NIPS'08)*. Curran Associates Inc., Red Hook, NY, 1497–1504.

[85] Özgür Şimşek, Alicia P. Wolfe, and Andrew G. Barto. 2005. Identifying useful subgoals in reinforcement learning by local graph partitioning (*ICML'05*). Association for Computing Machinery, New York, NY, 816–823. DOI: https://doi.org/10.1145/1102351.1102454

[86] Sungryull Sohn, Junhyuk Oh, and Honglak Lee. 2018. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates Inc., Red Hook, NY, 7156–7166.

[87] Martin Stolle and Doina Precup. 2002. Learning options in reinforcement learning. In *Proceedings of the International Symposium on Abstraction, Reformulation, and Approximation*. Springer, 212–223. DOI: https://doi.org/10.1007/3-540-45622-8_16

[88] Sainbayar Sukhbaatar, Emily Denton, Arthur Szlam, and Rob Fergus. 2018. Learning goal embeddings via self-play for hierarchical reinforcement learning. arxiv:1811.09083 (2018).

[89] Sainbayar Sukhbaatar, Ilya Kostrikov, Arthur Szlam, and Rob Fergus. 2017. Intrinsic motivation and automatic curricula via asymmetric self-play. arxiv:1703.05407 (2017).

[90] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. 2016. Learning multiagent communication with backpropagation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*. Curran Associates Inc., Red Hook, NY, 2252–2260.

[91] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction (2nd ed.)* The MIT Press, Cambridge, MA.

[92] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems (NIPS'99)*. The MIT Press, Cambridge, MA, 1057–1063.

[93] Richard S. Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.* 112, 1–2 (Aug. 1999), 181–211. DOI: https://doi.org/10.1016/S0004-3702(99)00052-1

[94] Ming Tan. 1997. *Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 487–494.

[95] Hongyao Tang, Jianye Hao, Tangjie Lv, Yingfeng Chen, Zongzhang Zhang, Hangtian Jia, Chunxu Ren, Yan Zheng, Changjie Fan, and Li Wang. 2018. Hierarchical deep multiagent reinforcement learning. arxiv:1809.09332 (2018).

[96] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J. Mankowitz, and Shie Mannor. 2017. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*. AAAI Press, 1553–1561.

[97] E. Todorov, T. Erez, and Y. Tassa. 2012. MuJoCo: A physics engine for model-based control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 5026–5033. DOI: https://doi.org/10.1109/IROS.2012.6386109

[98] Claudio Turchetti. 2004. *Stochastic Models of Neural Networks*. Vol. 102. IOS Press.

[99] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. 2017. FeUdal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*. JMLR.org, 3540–3549.

[100] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Mach. Learn.* 8, 3–4 (1992), 279–292. DOI: https://doi.org/10.1007/BF00992698

[101] Jiachen Yang, Igor Borovikov, and Hongyuan Zha. 2020. Hierarchical cooperative multi-agent reinforcement learning with skill discovery. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1566–1574.

[102] Z. Yang, K. Merrick, L. Jin, and H. A. Abbass. 2018. Hierarchical deep reinforcement learning for continuous action control. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 11 (2018), 5174–5184. DOI: https://doi.org/10.1109/TNNLS.2018.2805379

[103] Tom Zahavy, Avinatan Hasidim, Haim Kaplan, and Yishay Mansour. 2020. Planning in hierarchical reinforcement learning: Guarantees for using local policies. *Proc. Mach. Learn. Res.*, Vol. 117. PMLR, 906–934.

[104] Shangtong Zhang and Shimon Whiteson. 2019. DAC: The double actor-critic architecture for learning options. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc., 2012–2022.

Received July 2020; revised February 2021; accepted February 2021