12-2022

End-to-end hierarchical reinforcement learning with integrated subgoal discovery

Shubham PATERIA

Budhitama SUBAGDJA
Singapore Management University, budhitamas@smu.edu.sg

Ah-hwee TAN
Singapore Management University, ahtan@smu.edu.sg

Chai QUEK

Citation

1

End-to-End Hierarchical Reinforcement Learning with Integrated Subgoal Discovery

Shubham Pateria, Budhitama Subagdja, Ah-Hwee Tan, *Senior Member, IEEE* and Chai Quek, *Senior Member, IEEE*

***Abstract*—Hierarchical Reinforcement Learning (HRL) is a promising approach to perform long-horizon goal-reaching tasks by decomposing the goals into subgoals. In a holistic HRL paradigm, an agent must autonomously discover such subgoals and also learn a hierarchy of policies that uses them to reach the goals. Recently introduced end-to-end HRL methods accomplish this by using the higher-level policy in the hierarchy to directly search the useful subgoals in a continuous subgoal space. However, learning such a policy may be challenging when the subgoal space is large. We propose LIDOSS, an end-to-end HRL method with an integrated subgoal discovery heuristic that reduces the search space of the higher-level policy, by explicitly focusing on the subgoals that have a greater probability of occurrence on various state-transition trajectories leading to the goal. We evaluate LIDOSS on a set of continuous control tasks in the MuJoCo domain against Hierarchical Actor Critic (HAC), a state-of-the-art end-to-end HRL method. The results show that LIDOSS attains better goal achievement rates than HAC in most of the tasks.**

***Index Terms*—Reinforcement Learning, Hierarchical Reinforcement Learning, subgoal discovery.**

I. Introduction

Hierarchical Reinforcement Learning (HRL) is a promising approach for performing long-horizon goal-reaching tasks, in which an agent needs to execute a long sequence of actions in a large state space to reach a goal state [1]–[7]. A particular form of HRL, referred to as Feudal HRL [4], [8], addresses the long-horizon goal-reaching problem by learning a hierarchy of policies that decompose the goal into a sequence of easily reachable subgoals. A subgoal can be a state in the original state space or in an abstract state space. In such a hierarchy, a higher-level policy learns to choose a sequence of subgoals, as its outputs, to reach the goal. Meanwhile, a lower-level policy learns to choose a sequence of actions to reach a subgoal chosen by the higher-level policy.

The main challenge in Fedual HRL is to train the hierarchy of policies in an end-to-end manner. This means that neither the subgoals are predefined nor the policies are pretrained. Recent approaches achieve this by directly using the subgoal space as the output space of the higher-level policy, where the subgoal space is a continuous vector space containing all possible subgoals [2], [3]. Therefore, the agent does not require predefined subgoals and can directly learn to choose the subgoals by training the higher-level policy through reward maximization. This, however, requires the higher-level policy to incrementally update its output over a large subgoal space, which requires a long training time.

This paper investigates the benefits of subsampling the output space of the higher-level policy to improve the rate at which the agent learns to reach the goals, in episodic goal-reaching tasks. In this regard, we propose a novel subgoal discovery heuristic which estimates the probability of occurrence (or simply, *occurrence probability*) of different subgoal states on various state-transition trajectories[1] leading to a goal during training, finds the subgoals with higher occurrence probabilities within local neighbourhoods in the subgoal space, and selects those subgoals as the salient subgoals. The salient subgoals are then used as the outputs of the higher-level policy. The intuition behind the proposed heuristic is that a subgoal with a higher occurrence probability recorded after multiple episodes is more likely to appear on various possible trajectories leading to the goal in the future episodes as well. Hence, such a subgoal can be chosen by the higher-level policy as an intermediate target to reach the goal.

The complete method, with the subgoal discovery heuristic integrated with the end-to-end training of the hierarchy of policies, is named Hierarchical Reinforcement **L**earning with **I**ntegrated **D**iscovery **O**f **S**alient **S**ubgoals (LIDOSS). We evaluate LIDOSS on a set of continuous control tasks in the MuJoCo domains [9], with fixed goals and dynamic goals; whereby its performance is compared with that of Hierarchical Actor Critic (HAC) [2], a state-of-the-art end-to-end HRL method which uses a continuous subgoal space as the output space of the higher-level policy without subgoal discovery. Additionally, we also compare LIDOSS with other heuristics for subsampling the output space of the higher-level policy. The results show that LIDOSS outperforms HAC and other heuristics in most of the tasks.

LIDOSS was first introduced in our previous work [10]. It is further improved in this paper as follows: Firstly, the previous implementation of LIDOSS used a Deep Q Network (DQN) [11] which took a state and a goal as the inputs, and predicted the Q-values of all subgoals that could be chosen by the higher-level policy. The outputs of the DQN (that is, the predicted Q-values) were independent of each other. On the other hand, the new implementation of LIDOSS presented in this paper uses a single-output Q network which

S. Pateria and C. Quek are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore, 639798 SG (e-mail: SHUBHAM007@e.ntu.edu.sg, ashcquek@ntu.edu.sg).

B. Subagdja and A.H. Tan are with the School of Computing and Information Systems, Singapore Management University, Singapore, 178902 SG (e-mail: budhitama@ntu.edu.sg).

[1] A state-transition trajectory is the sequence of states observed during an episode

takes a subgoal as an input, along with a state and a goal, and predicts the Q-value of that subgoal (refer to subsection IV-A). This change has been made because the single-output Q network approximates the higher-level Q-value function in the continuous subgoal space by taking a subgoal as one of the inputs; hence, it can better generalize the predicted Q-value to the previously unseen subgoals. This is important because the subgoals in LIDOSS are incrementally discovered. Such a generalization cannot be achieved using DQN which predicts independent Q-values of different subgoals.

Secondly, the previous implementation of LIDOSS only discovered the salient subgoals with respect to the exogenous goal specified as part of the goal-reaching task. If the agent did not reach that goal in an episode, the episodic data was not used for subgoal discovery. The new implementation of LIDOSS uses the episodic data more prudently by treating the terminal state reached in an episode as a *proxy* goal and discovering different sets of subgoals with respect to the different proxy goals encountered (refer to subsection IV-B). This is especially favourable in a task with dynamically changing goals (refer to section V) because the subgoals discovered with respect to a proxy goal encountered in the earlier episodes can be useful if the exogenous goal in a future episode is the same as that proxy goal.

The new implementation of LIDOSS shows better performance than the previous implementation. The ablative comparison of the these implementations is provided in Appendix *E* of the Supplementary Material.

The rest of the paper is organized as follows: The relevant related work is covered in section II. The essential preliminary concepts are discussed in section III. Section IV describes the proposed method LIDOSS. This is followed by the discussion of experiments and results in section V. Finally, the paper is concluded in section VI.

II. Related Work

Most modern Feudal HRL methods are based on the tabular Q-learning 'manager' and 'sub-manager' hierarchy conceptualized by Dayan et al. [8]. In this concept, the policy at each level in the hierarchy is regarded as the 'manager' which chooses the subgoals for the lower-level policy as well as the 'sub-manager' which achieves the subgoals given by the higher-level policy (refer to subsection III-B). Vezhnevets et al. [4] extended this concept and proposed an end-to-end Feudal HRL method comprising of a hierarchy of deep neural networks, for learning in high-dimensional state spaces. A higher-level network chooses a subgoal vector from a learned latent subgoal space. A lower-level network learns to take the actions to achieve the subgoal. This method, called Feudal Networks (FuN), shows better performance than a non-hierarchical agent. However, FuN requires extensive training to learn the latent subgoal space. Nachum et al. [3] relax this requirement by directly using the state space of the agent as the subgoal space in a method called HIerarchical Reinforcement learning with Off-policy correction (HIRO). But the more significant contribution of this work is an off-policy correction method to deal with the issue of non-stationarity in the higher-level Q-value updates encountered when multiple policies in the hierarchy are being trained simultaneously. Concurrently, Levy et al. [2] proposed a method called Hierarchical Actor Critic (HAC) which addresses the similar non-stationarity issue, uses the concept of Hindsight Experience Replay (HER) [12] to learn the HRL agent with sparse rewards, and also provides a subgoal testing procedure to encourage a higher-level policy to choose reachable subgoals for a lower-level policy. Due to these technical features, HAC is shown to outperform HIRO on the continuous control benchmarks. FuN [4], HIRO [3], and HAC [2] are important contributions towards the goal of developing end-to-end Feudal HRL. However, they might require long periods of training since the highest-level policy needs to adjust its output over a large continuous subgoal space. In contrast, LIDOSS simplifies the output space of the highest-level policy by using the discrete set of subgoals discovered by the integrated subgoal discovery heuristic.

Several subgoal discovery heuristics also exist in the HRL literature, such as the count-based statistics [13], [14], graph-based clustering [15]–[17], spectral analysis [18], [19] etc. These approaches perform off-line subgoal discovery which is separated from the process of learning the HRL agent's hierarchy of policies. This causes an overhead because the agent cannot start learning to perform a task without going through a separate phase of subgoal discovery. In contrast, LIDOSS trains an agent in an end-to-end manner via integration of the subgoal discovery heuristic and the learning of the hierarchy of policies.

Lastly, hierarchical learning paradigms also exist in the area of unsupervised clustering in the form of *hierarchical clustering* [20], [21], and in the area of supervised classification in the form of *hierarchical classification* [22], [23]. *Hierarchical clustering* develops a hierarchy of clusters by either repeatedly merging small clusters into a larger one (agglomeration strategy) or splitting a larger cluster into smaller ones (divisive strategy) [21]. Yildirim et al. [21] proposed a k-Linkage agglomeration strategy which evaluates the distance between two lower-level clusters by calculating the average distance between k pairs of observations, one from each cluster. The lower-level clusters are merged into the higher-level ones based on these calculated distances. This k-Linkage approach is shown to yield more stable hierarchical clusters than other standard linkage methods for inter-cluster distance estimations. *Hierarchical classification* is the process of classifying a given data into classes that are organized into a hierarchy. A limitation of the standard hierarchical classification is that the exact hierarchy of the class labels needs to be predefined for the data instances, which makes incremental classification hard. Recently, Park et al. [23] addressed this problem of incremental hierarchical classification using hierarchically stacked modules of fuzzy Adaptive Resonance Theory-supervised Predictive Mapping (ARTMAP) [15]. The module at each level assigns the class at that level to a new data instance and also decides whether to perform classification at the higher level. In this way, the stacked ARTMAP performs incremental hierarchical classification.

Notation	Definition
$\mathbb{S}$	State space
$\mathbb{A}$	Primitive action space
$\mathbb{G}$	Goal space
G	General notation for a goal in the goal space $\mathbb{G}$
$\mathbb{G}^{sub}$	Subgoal space
g	General notation for a subgoal in the subgoal space $\mathbb{G}^{sub}$
ϕ	Abstraction function for mapping a state to a goal, that is, $G = \phi(s)$; where $s \in \mathbb{S}$
ϕ^{sub}	Abstraction function for mapping a state to a subgoal, that is, $g = \phi^{sub}(s)$; where $s \in \mathbb{S}$
π^{HL}	Highest-level policy of a general Feudal HRL agent
$\pi\epsilon^{HL}$	Highest-level policy of the LIDOSS agent, which is an ϵ-greedy policy
π^{HL-1}	Intermediate-level policy of a general Feudal HRL agent (same for LIDOSS)
π^{LL}	Lowest-level policy of a general Feudal HRL agent (same for LIDOSS)
$\mathcal{G}^{sub}_{SDM}(G)$	Discrete salient subgoal set discovered by LIDOSS, containing subgoals of G
g^{HL}	Subgoal chosen by π^{HL} from $\mathbb{G}^{sub}$ (in general) or by $\pi\epsilon^{HL}$ from $\mathcal{G}^{sub}_{SDM}(G)$ (in case of LIDOSS)
g^{HL-1}	Subgoal chosen by π^{HL-1} from $\mathbb{G}^{sub}$
Q^{HL}	Highest-level Q-value function of a general Feudal HRL agent (same for LIDOSS)
Q^{HL-1}	Intermediate-level Q-value function of a general Feudal HRL agent (same for LIDOSS)
Q^{LL}	Lowest-level Q-value function of a general Feudal HRL agent (same for LIDOSS)
R_{in}	Internal reward used by a general Feudal HRL agent (same for LIDOSS) to learn Q^{HL-1}, Q^{LL}, π^{HL-1}, and π^{LL}
c^{HL}	Time period for which a highest-level subgoal g^{HL} is repeated before choosing a new one
c^{HL-1}	Time period for which an intermediate-level subgoal g^{HL-1} is repeated before choosing a new one
q	The quantization size for the subgoal space quantization in the subgoal discovery heuristic of LIDOSS (in SDM; subsection IV-B)
q^G	The quantization size for the goal space quantization in the subgoal discovery heuristic of LIDOSS (in SDM)
$\lvert LMX \rvert$	The size of Local Max (LMX) kernel used by the subgoal discovery heuristic of LIDOSS (in SDM)
ψ_{th}	The threshold *occurrence probability* of a quantized subgoal within LMX for it to be considered as salient subgoal (in SDM)
SRR	SDM Refresh Rate. The SDM updates the salient subgoal sets after every SRR^{th} episode

TABLE I: Notations and their definitions.

III. Preliminaries

The concepts outlined in the following subsections III-A, III-B, and III-C are crucial to understand the design of our method LIDOSS (in section IV) and the results of the experiments (in section V). A list of notations used in various sections of the paper is provided in Table I.

A. Universal Markov Decision Process

We consider a problem setting in which an agent is given a goal that should be reached within a limited amount of time. Such a problem can be specified to the agent in the form of a Universal Markov Decision Process (UMDP) [2]. A UMDP consists of a state space $\mathbb{S}$, an action space $\mathbb{A}$, a goal space $\mathbb{G}$, a reward function $R(s, a|G)$ defined with respect to a goal $G \in \mathbb{G}$, a state $s \in \mathbb{S}$, and an action $a \in \mathbb{A}$. The solution of a UMDP is a goal-conditioned policy $\pi(a|s, G)$: $\mathbb{S} \times \mathbb{G} \to \mathbb{A}$ which takes a state and a goal as inputs and chooses an action $a \in \mathbb{A}$ as the output. This policy is obtained by maximizing a goal-conditioned Q-value function [24] which is defined as follows, $Q^{\pi(a|s,G)}(s, a|G) = \mathbb{E}_{a_t \sim \pi(a_t|s_t,G)}[\sum_{t=0}^{t=T^s_G} \gamma^t R(s_t, a_t|G)|s_0{=}s, a_0{=}a]$, where $\gamma \in (0, 1]$ is a discount factor and T^s_G is the time-horizon over which G is reached from the start state s. We use T_G to denote the average time-horizon over which G is achieved from different start states.

In this paper, the goal space $\mathbb{G}$ is defined as a lower-dimensional abstraction of the state space $\mathbb{S}$, obtained using a predefined domain-specific[2] abstraction function $\phi(s)$. This function can map a set of states to the same goal. Technically, *a goal represents a set of target states, any of which the agent needs to reach.*

All the components of the UMDP discussed above, including the goal, are externally specified to the agent as part of the goal-reaching problem. Henceforth, we refer to the externally specified UMDP as the ***core*** UMDP. The externally specified goal is referred to as ***exogenous*** goal *wherever it is necessary to* clearly distinguish it from any goal or subgoal internally chosen by the agent for learning or subgoal discovery, discussed later.

B. Feudal HRL and Temporal Abstraction

In the scope of this paper, Feudal HRL [2], [8] can be formally defined in terms of a hierarchy of UMDPs as follows:

Given a *core* UMDP, the Feudal HRL agent internally decomposes it into a hierarchy of subgoal-based UMDPs. The agent has two internal components: a subgoal space $\mathbb{G}^{sub}$ and a reward function $R_{in}(s, a|g)$ defined with respect to a subgoal $g \in \mathbb{G}^{sub}$. Similar to the goal space, the subgoal space is

[2]specific to a particular task domain (refer to section V)

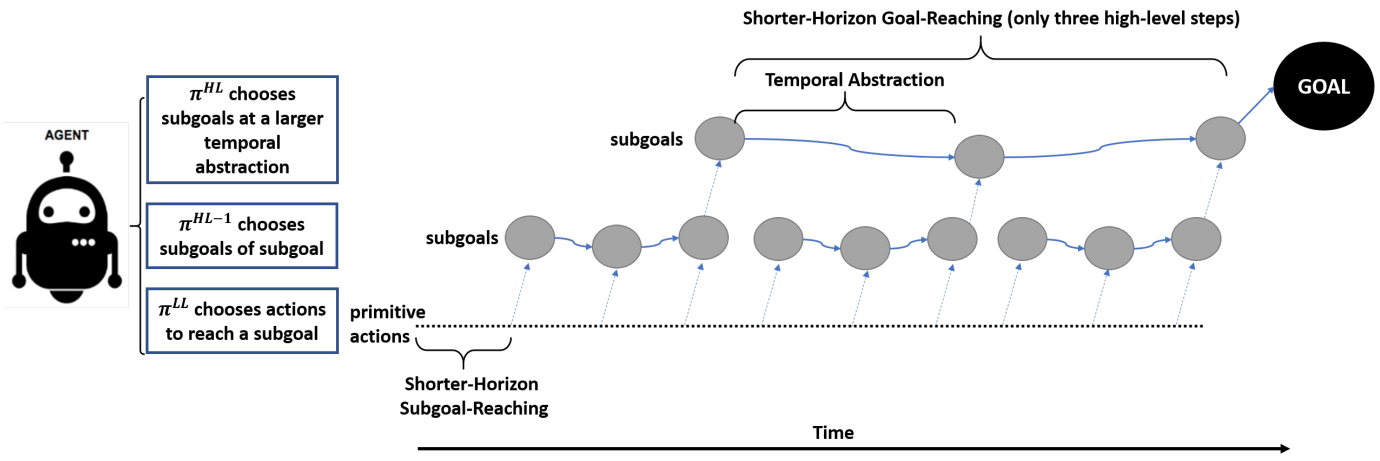

AGENT
π^{HL} chooses subgoals at a larger temporal abstraction
π^{HL-1} chooses subgoals of subgoal
π^{LL} chooses actions to reach a subgoal
Shorter-Horizon Goal-Reaching (only three high-level steps)
Temporal Abstraction
GOAL
subgoals
subgoals
primitive actions
Shorter-Horizon Subgoal-Reaching
Time

Fig. 1: An example of Feudal HRL, with larger temporal abstraction at the highest level and shorter-horizon (reachable) subgoals at the lowest level.

also defined as a lower-dimensional abstraction of the state space. A state $s \in \mathbb{S}$ can be mapped to a corresponding lower-dimensional subgoal in $\mathbb{G}^{sub}$, via a predefined domain-specific abstraction function $\phi^{sub}(s)$. The abstraction function can map a set of states to the same subgoal. Technically, *a subgoal represents a set of intermediate states between the current state and the goal*. The internal reward $R_{in}(s, a|g)$ is domain-specific, but generally defined as greater than or equal to zero only in the states near the subgoal, otherwise it has a negative value [2].

The highest-level UMDP in the hierarchy is denoted as $l = HL$. Its state space and goal space are the same as those of the core UMDP. However, its action space is the subgoal space $\mathbb{G}^{sub}$. The highest-level policy, denoted as $\pi^{HL}(g^{HL}|s, G)$, takes a state $s \in \mathbb{S}$ and an exogenous goal $G \in \mathbb{G}$ as the inputs, and chooses a subgoal $g^{HL} \in \mathbb{G}^{sub}$ as its action[3]. The chosen subgoal is repeated for c^{HL} time steps or until it is reached, before a new subgoal can be chosen (refer to subsection III-C to understand how this is implemented). c^{HL} is greater than one but less than T_G. Henceforth, we refer to the chosen subgoal as the *output* of π^{HL} rather than *action*, to avoid confusion with the per-step action taken by the agent in the core UMDP. The reward used to train the highest-level policy is defined as $R^{HL}(s_t, g^{HL}|G) = \sum_{i=0}^{i=c^{HL}} \gamma^i R(s_{t+i}, a_{t+i}|G)$. Here, $R(s, a|G)$ is the reward function of the core UMDP (subsection III-A) and $a_t \in \mathbb{A}$ is an action taken by the agent in the core UMDP at time t, using the lowest-level policy described below. The Q-value is denoted as $Q^{HL}(s, g^{HL}|G)$.

The intermediate-level UMDP in the hierarchy is denoted as $l = HL - 1$. Its state space is the same as that of the core UMDP. However, both the goal space and the action space are same as the subgoal space $\mathbb{G}^{sub}$. The policy at this level, denoted as $\pi^{HL-1}(g^{HL-1}|s, g^{HL})$, takes the output subgoal g^{HL} of the highest-level policy π^{HL} as an input along with the state $s \in \mathbb{S}$, and chooses a subgoal $g^{HL-1} \in \mathbb{G}^{sub}$ as the action. Henceforth, we refer to the chosen subgoal as the *output* of π^{HL-1} rather than the *action*. The output g^{HL-1} is basically *a subgoal of the highest-level subgoal*. The chosen subgoal is repeated for c^{HL-1} time steps or until it is reached, before a new subgoal can be chosen. c^{HL-1} is greater than one but less than c^{HL}. The reward used to train the intermediate-level policy is defined as $R^{HL-1}(s_t, g^{HL-1}|g^{HL}) = \sum_{i=0}^{i=c^{HL-1}} \gamma^i R_{in}(s_{t+i}, a_{t+i}|g^{HL})$, where $R_{in}(s, a|g^{HL})$ is the internal reward function of the agent, defined earlier. The Q-value is denoted as $Q^{HL-1}(s, g^{HL-1}|g^{HL})$.

The lowest-level UMDP in the hierarchy is denoted as $l = LL$. Its state space and the action space are the same as those of the core UMDP. However, its goal space is the subgoal space $\mathbb{G}^{sub}$. The policy at this level, denoted as $\pi^{LL}(a|s, g^{HL-1})$, takes the output subgoal $g^{HL-1} \in \mathbb{G}^{sub}$ of the intermediate-level policy as an input along with the state $s \in \mathbb{S}$, and chooses an action $a \in \mathbb{A}$. Henceforth, we refer to a as a *primitive* action. The repetition time of the chosen action is $c^{LL} = 1$, that is, the agent takes a primitive action at every time step. The reward used to train the lowest-level policy is $R^{LL}(s_t, a_t|g^{HL-1}) = R_{in}(s_t, a_t|g^{HL-1})$. The Q-value is denoted as $Q^{LL}(s, a|g^{HL-1})$.

1) Temporal Abstraction in Feudal HRL: When the exogenous goal is distant from all the start states in a core UMDP, it poses the long-horizon goal-reaching problem which can be difficult to solve using standard reinforcement learning approaches [4], [25]. Feudal HRL addresses this problem through *temporal abstraction* [26] achieved using the subgoals. Temporal abstraction at the highest level of the hierarchy means that each *individual* decision step of the policy π^{HL} (that is, choosing a subgoal g^{HL}) occurs after multiple primitive action steps (equal to c^{HL} steps). This reduces the long-horizon goal-reaching problem into a shorter-horizon problem at the highest level, since the policy π^{HL} needs to learn a sequence of subgoals that is much shorter than the sequence of primitive actions (see the example in Fig. 1). At the same time, the intermediate-level policy π^{HL-1} takes the highest-level subgoal as the target and chooses a sequence of subgoals, which provides temporal abstraction at the intermediate level (Fig. 1). Lastly, the

[3]The general notation for a subgoal is $g \in \mathbb{G}^{sub}$. The superscript in g^{HL} is used to clarify that the subgoal is chosen by the highest-level policy.

lowest-level policy π^{LL} takes an intermediate-level subgoal as the target to choose the primitive actions, which poses a shorter-horizon learning problem at the lowest level (Fig. 1). This hierarchical decomposition of a long-horizon goal-reaching problem into the shorter-horizon problems can potentially improve the learning of the goal-reaching behaviour [2]–[4].

2) Trade-off between Temporal Abstraction and Subgoal Reachability: The benefit of temporal abstraction could be obtained using a two-level hierarchy with just π^{HL} and π^{LL}. However, larger temporal abstraction at the highest level means that the lowest-level policy π^{LL} must take a longer sequence of primitive actions to reach a subgoal. Such a subgoal might become unreachable within a limited time when the lowest-level policy is untrained. On the other hand, the intermediate-level policy π^{HL-1} in a three-level hierarchy might resolve this trade-off. In this case, the highest-level policy π^{HL} can choose a subgoal g^{HL} at a larger temporal abstraction but that subgoal is not given to the lowest-level policy π^{LL} as an input. Rather, π^{HL-1} takes that subgoal as input and chooses a *subgoal of subgoal* g^{HL-1}, which requires fewer primitive actions to be reached using π^{LL} (see Fig. 1). Hence, subgoal reachability is maintained at the lowest level. This trade-off between temporal abstraction and subgoal reachability is important for analyzing the construction of our approach (section IV) and the results of the experiments (section V).

C. Base Framework: Hierarchical Actor Critic (HAC)

Hierarchical Actor Critic (HAC) [2] is a state-of-the-art framework for end-to-end training of a Feudal HRL agent. HAC provides both temporal abstraction and subgoal reachability, as discussed further below. It uses an actor-critic [27] combination at each level of the hierarchy of policies. The critic at level l is a Q-value function Q^l which is learned by *minimizing* the following squared error [27],

$$\left(Q^l(s_i, g^l|g^{l+1}; \theta^{Q^l}) - \left[R^l + \gamma^{c^l} Q^l(s_{i+c^l}, \hat{g}^l|g^{l+1}; \theta^{Q^l}\right]\right)^2 \tag{1}$$

where $\hat{g}^l$ is the next subgoal chosen by π^l and θ^{Q^l} are the parameters of the Q-value function. The actor is a policy π^l which is learned using the following policy gradient [27],

$$\nabla_{g^l} Q^l(s_i, g^l|g^{l+1}) \nabla_{\theta^{\pi^l}} \pi^l(g^l|s_i, g^{l+1}; \theta^{\pi^l}) \tag{2}$$

Here, g^l is the output of π^l in a state s_i and θ^{π^l} are the parameters of the policy π^l. For $l = HL$, $g^{l+1} = G \in \mathbb{G}$, that is, the exogenous goal. For $l = LL$, the output $g^l = a \in \mathbb{A}$, that is, the primitive action. Equation 2 implies that the output of the policy π^l is incrementally shifted in the continuous subgoal space until it converges on a region where the Q-value is maximum and the gradient $\nabla_{g^l} Q^l(s_i, g^l|g^{l+1})$ tends to zero.

How does HAC achieve temporal abstraction? As per equation 2, the actor at each level (above the lowest-level) is trained to choose a subgoal that has a larger Q-value, from the continuous subgoal space. In a continuous subgoal space, the subgoals closer to the goal have a higher Q-value as compared to the subgoals closer to the current state, especially if the rewards are greater than or equal to zero *only near* the goal states. This naturally encourages the actor to choose the subgoals which are farther from the current state and closer to the goal, hence requiring multiple primitive actions to reach and achieving temporal abstraction.

How does HAC maintain subgoal reachability? HAC maintains subgoal reachability via a procedure called *subgoal testing*. HAC tests if a subgoal g^l chosen by a higher-level policy π^l is achieved within c^l time steps or not, where c^l is the designated subgoal repetition time. If the subgoal is not achieved within c^l time steps, the policy π^l receives a *penalty* (negative reward) proportional to c^l, for choosing that subgoal. This encourages π^l to learn to choose those subgoals which can be reached within c^l time steps.

Other features of HAC are discussed in Appendix *A* of the Supplementary Material. Our method, discussed in section IV, uses HAC as the base framework. Therefore, it uses the above-mentioned features of HAC.

IV. LIDOSS: HRL with Integrated Discovery of Salient Subgoals

This section presents the details of our method LIDOSS, including the composition of the LIDOSS agent (in subsection IV-A) and the heuristic used for subgoal discovery (in subsection IV-B). A list of advantages and limitations of LIDOSS is provided in the Conclusion (section VI) of this paper.

A. LIDOSS Agent Composition

The LIDOSS agent shares the following components with the HAC framework: The lower-level policies (actors) π^{HL-1} and π^{LL}, the lower-level Q-value functions (critics) Q^{HL-1} and Q^{LL}, their input space, output space, and the learning rules (equation 1 and 2) are the same as in HAC. LIDOSS also uses the *subgoal testing* (subsection III-C) at all the levels to learn to choose reachable subgoals.

LIDOSS differs from HAC by integrating the subgoal discovery at the highest level, using a non-parametric discrete-action policy as the highest-level policy, and using the discrete set of discovered subgoals as the outputs of that highest-level policy. These aspects of LIDOSS are as described below.

In HAC, the highest-level policy π^{HL} incrementally shifts its output over the continuous subgoal space via the policy parameter-update ($\nabla_{\theta^{\pi^{HL}}}$) based on the gradient of the Q-value of the current output ($\nabla_{g^{HL}} Q^{HL}(s_i, g^{HL}|G)$) (equation 2). This can be interpreted as the highest-level policy searching for an optimal subgoal over the large continuous subgoal space. On the other hand, LIDOSS uses a Subgoal Discovery Module (SDM) to discover a *sparse and discrete* set of salient subgoals from the continuous subgoal space $\mathbb{G}^{sub}$, which can be used as the *discrete* outputs of the highest-level policy. This effectively reduces the search space of the highest-level policy by restraining it to the discovered salient subgoals.

Furthermore, LIDOSS uses a non-parametric discrete-action $\epsilon - greedy$ policy at the highest level, denoted as

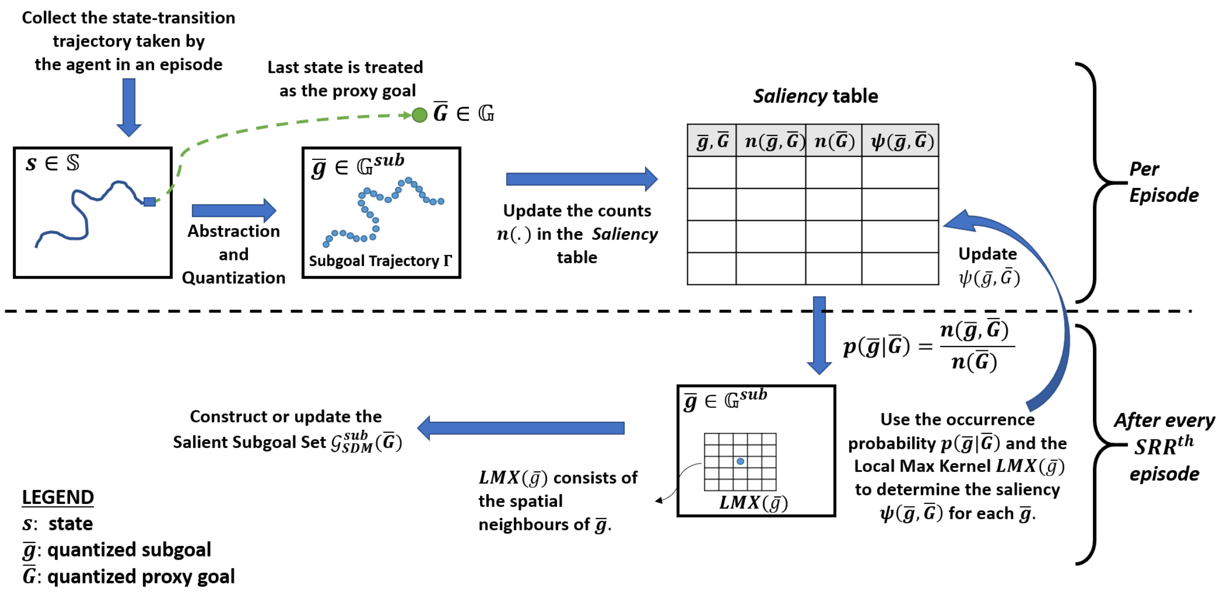

Collect the state-transition trajectory taken by the agent in an episode
Last state is treated as the proxy goal
$\bar{G} \in \mathbb{G}$
$s \in \mathbb{S}$
Abstraction and Quantization
$\bar{g} \in \mathbb{G}^{sub}$
Subgoal Trajectory Γ
Update the counts $n(.)$ in the *Saliency* table
Saliency table
$\bar{g}, \bar{G}$
$n(\bar{g}, \bar{G})$
$n(\bar{G})$
$\psi(\bar{g}, \bar{G})$
Update $\psi(\bar{g}, \bar{G})$
Per Episode
$p(\bar{g}|\bar{G}) = \frac{n(\bar{g}, \bar{G})}{n(\bar{G})}$
$\bar{g} \in \mathbb{G}^{sub}$
$LMX(\bar{g})$
Use the occurrence probability $p(\bar{g}|\bar{G})$ and the Local Max Kernel $LMX(\bar{g})$ to determine the saliency $\psi(\bar{g}, \bar{G})$ for each $\bar{g}$.
After every SRR^th episode
Construct or update the Salient Subgoal Set $\mathcal{G}^{sub}_{SDM}(\bar{G})$
$LMX(\bar{g})$ consists of the spatial neighbours of $\bar{g}$.
LEGEND
s: state
$\bar{g}$: quantized subgoal
$\bar{G}$: quantized proxy goal

Fig. 2: The flow of operations in the Subgoal Discovery Module (SDM).

$\pi\epsilon^{HL}(g^{HL}|s, G)$. It chooses the subgoal with the largest Q-value with probability $1-\epsilon$ and chooses a random subgoal with probability ϵ (for exploration), from the set of the discovered salient subgoals. This also reduces the overhead of learning a parametric continuous-action policy, such as in HAC.

The Q-value functions at all levels, including the highest level, are implemented using single-output Q networks. The highest-level Q network takes a subgoal as an input along with a state and exogenous goal, and predicts the Q-value of that subgoal (as also mentioned in section I). The learning rule for the highest-level Q-value function is same as in equation 1, with the subgoals chosen by $\pi\epsilon^{HL}$.

SDM is described in subsection IV-B. It *incrementally* extracts the salient subgoals from the continuous subgoal space $\mathbb{G}^{sub}$ by using a probability-based subgoal discovery heuristic. The discrete set of salient subgoals corresponding to an exogenous goal G is denoted as $\mathcal{G}^{sub}_{SDM}(G)$. At the beginning of each episode, the LIDOSS agent fetches the subgoal set $\mathcal{G}^{sub}_{SDM}(G)$ from SDM, for the exogenous goal G given in that episode. The goal itself is added to this set, making the set of outputs of $\pi\epsilon^{HL}$ equal to $\mathcal{G}^{sub}_{SDM}(G)$ + $\{G\}$. Thus, if the salient subgoals are not available initially, the intermediate-level policy π^{HL-1} takes the exogenous goal as the input subgoal. This usually happens in the initial stages of training since the subgoal discovery in SDM is an incremental process. When the salient subgoals are available in $\mathcal{G}^{sub}_{SDM}(G)$, the policy $\pi\epsilon^{HL}$ chooses a subgoal from $\mathcal{G}^{sub}_{SDM}(G)$ + $\{G\}$, which is then given as the input to π^{HL-1}.

In the next subsection, we discuss how the SDM discovers a set of salient subgoals.

B. LIDOSS Subgoal Discovery

The subgoal discovery heuristic used in LIDOSS searches for the subgoal states that appear more frequently than others on the state-transition trajectories[4] connecting various start states to the goal(s). It selects such subgoals as the *salient subgoals*, which are then used as the outputs of the highest-level policy of the LIDOSS agent (subsection IV-A). The frequency of occurrence of any subgoal g on the trajectories leading to a goal G is represented by an *occurrence probability* $p(g|G)$, which is estimated using the episodic data gathered by the agent during exploration and training. The intuition behind the proposed heuristic is based on the following assumption,

Assumption 1: A subgoal g with a greater occurrence probability $p(g|G)$ recorded after multiple episodes is more likely to appear on various possible trajectories leading to the goal G in the future episodes as well. Hence, it is one of the *salient subgoals* and it can be used as one of the outputs of the highest-level policy of LIDOSS agent. That policy can choose g as an intermediate target to reach the goal in the future. This assumption also holds if the behaviour of the agent changes in the future as a result of training or stochasticity, leading to the discovery of new salient subgoals. In such a case, g can still be retained in the salient subgoal set without directly affecting the highest-level policy, because that policy chooses an output from the salient subgoal set *based on the latest Q-values* (refer to subsection IV-A).

The proposed subgoal discovery heuristic is designed to achieve the following properties without using the rewards or the subgoal testing penalty (subsection III-C) that are used to train the policies:

Bottleneck Subgoal Discovery: Bottlenecks are those states in the state space where several trajectories leading to the goal(s) converge [13], [28]. The implication is that the topol-

[4]A state-transition trajectory is the sequence of states observed during an episode

ogy of the state space forces the agent to go through the bottleneck states to reach the goal(s). Such bottlenecks increase the difficulty of reaching the long-horizon goal(s). The probability-based subgoal discovery in LIDOSS is intended to find the subgoals corresponding to such bottleneck states (remember that a subgoal is an abstraction of a state; subsection III-B).

Subgoal Reachability: As discussed in subsection III-B, subgoal reachability is important in the end-to-end HRL. Hence, the subgoal discovery heuristic must select the salient subgoals in such a manner that the subgoal reachability is maintained. This is achieved by comparing the $p(g|G)$ values of different subgoals within local neighbourhoods in the subgoal space, rather than comparing them globally. This is called Local Max Saliency determination, described later in subsection IV-B2. Selecting the subgoals with globally larger $p(g|G)$ might result in most of the discovered subgoals being concentrated only around the bottlenecks. If the bottleneck subgoals are themselves difficult to reach during the early stages of training, then the performance of the agent might suffer. The heuristic of LIDOSS selects at most one subgoal from each local neighbourhood as the salient subgoal through the local comparison of $p(g|G)$ values. This ensures that the discovered salient subgoals are not over-concentrated in a single neighbourhood, such as around the bottlenecks, but more widely distributed over the subgoal space to maintain the local reachability of the subgoals.

The subgoal discovery heuristic is further explained below. A corresponding pseudo code is provided later in Algorithm 1.

1) Procedure of Subgoal Discovery: The procedure of subgoal discovery involves multiple stages of operations carried out *after every episode*. The operations are depicted as part of the Subgoal Discovery Module (SDM) in Fig. 2. Firstly, the state-transition trajectory taken by the agent during an episode is received by the SDM. The terminal state in this trajectory is abstracted as a *proxy* goal $G \in \mathbb{G}$ using the pre-defined goal space abstraction function $\phi(s)$. Using the terminal states as the *proxy* goals helps the SDM discover multiple sets of subgoals corresponding to a variety of goals and also avoid wasting an episodic data if the exogenous goal is not achieved in that episode. Furthermore, each state in the trajectory is mapped to a corresponding subgoal in the subgoal space $\mathbb{G}^{sub}$ via the predefined subgoal abstraction function $\phi^{sub}(s)$. The trajectory consisting of the subgoals is denoted as Γ_g.

Next, the SDM quantizes the proxy goal and the subgoals present in Γ_g. Quantization is performed to enable the counting (across multiple episodes) of the occurrence of the proxy goal, and the occurrence of subgoals on the trajectories leading to the proxy goal. In a continuous space, the exactly same proxy goal or subgoal may not be encountered twice in different episodes [29]. This could make the count of every subgoal and proxy goal equal to one, leading to equal occurrence probabilities and failure of subgoal discovery. Quantization ensures that the occurrences of the subgoals (or proxy goals) which are very close to each other are added into the occurrence of a single quantized subgoal (or quantized proxy goal). The quantization is performed as follows:

$$\bar{g} = \lfloor \frac{g}{q} \rfloor \times q \tag{3}$$

for the subgoal, and

$$\bar{G} = \lfloor \frac{G}{q^G} \rfloor \times q^G \tag{4}$$

for the goal. Here, $\bar{g}$ and $\bar{G}$ are the quantized subgoal and proxy goal, respectively. q and q^G are the quantization sizes. The notation $\lfloor \cdot \rfloor$ indicates the *floor* function. The quantization sizes are kept very small such that the quantized subgoals and proxy goals do not violate the topology of the subgoal space $\mathbb{G}^{sub}$ and the goal space $\mathbb{G}$, respectively (refer to Appendix *C* of the Supplementary Material). The new quantized subgoal trajectory is denoted as $\Gamma'_{\bar{g}}$.

After quantization, SDM updates a *Saliency Table* for the entries corresponding to different pairs $(\bar{g}, \bar{G})$. The *Saliency Table* consists of the following entries: (i) *($\bar{g}$, $\bar{G}$) pair*, which is the *key* of a row. (ii) $n(\bar{G})$, which is the count of occurrence of a quantized proxy goal $\bar{G}$, incremented over multiple episodes. (iii) $n(\bar{g}, \bar{G})$, which is the count of occurrence of a quantized subgoal $\bar{g}$ on the trajectories terminating in the proxy goal $\bar{G}$, incremented over multiple episodes. (iv) $\psi(\bar{g}, \bar{G})$, which is the saliency value of $\bar{g}$ with respect to $\bar{G}$. Then, **the occurrence probability is calculated as** $p(\bar{g}|\bar{G}) = \frac{n(\bar{g},\bar{G})}{n(\bar{G})}$.

In this manner, SDM maintains the counts of the quantized subgoals and quantized proxy goals over multiple episodes. SDM updates the salient subgoal sets, denoted as $\mathcal{G}^{sub}_{SDM}(\bar{G})$, corresponding to different quantized proxy goals stored in the *Saliency Table*, after every SRR^{th} episode. Here, SRR stands for SDM Refresh Rate. The update procedure is described in the following subsections IV-B2 and IV-B3.

2) Local Max Saliency: As mentioned earlier, the subgoal discovery heuristic selects the salient subgoals within local neighbourhoods such that their reachability is maintained. SDM determines the saliency of a quantized subgoal by comparing its probability values with its local neighbour quantized subgoals. Local comparisons ensure that if the occurrence probability of a quantized subgoal is higher than its neighbours and passes a threshold, it is selected as a salient subgoal even if it does not have a globally higher occurrence probability.

The saliency of a quantized subgoal is determined using equation 5, defined below. In this equation, $LMX(\bar{g})$ is a Local Max kernel which consists of the neighbouring quantized subgoals around $\bar{g}$ (Fig. 2). The number of neighbours in this kernel is denoted as $|LMX|$. We use a spatial neighbourhood to constitute LMX. Then, $\max_{LMX(\bar{g})} = \max_{\bar{g}' \in LMX(\bar{g})} p(\bar{g}'|\bar{G})$ is the maximum occurence probability among the neighbours of $\bar{g}$, with respect to a proxy goal $\bar{G}$. If the saliency $\psi(\bar{g}, \bar{G})$=1, then $\bar{g}$ is considered as a salient subgoal with respect to $\bar{G}$.

$$\psi(\bar{g}, \bar{G}) = \begin{cases} 0, & \text{if } p(\bar{g}|\bar{G}) < \max_{LMX(\bar{g})} \\ 0, & \text{if } \max_{LMX(\bar{g})} <= \\ & \quad p(\bar{g}|\bar{G}) < \psi_{th} \\ 1, & \text{otherwise} \end{cases} \tag{5}$$

Equation 5 implies that a quantized subgoal $\bar{g}$ is considered salient with respect to a quantized proxy goal $\bar{G}$ only if it passes two conditions. The first condition is that the probability $p(\bar{g}|\bar{G})$ should not be less than the maximum probability among the neighbouring quantized subgoals (that is, $\max_{LMX(\bar{s})}$). The second condition is that even if $p(\bar{g}|\bar{G})$ is greater than or equal to the maximum probability among the neighbours, it should not be less than ψ_{th}, which is a threshold probability value called *saliency threshold*. If there are multiple states within $LMX(\bar{g})$ with non-zero saliency, only one of them is chosen as the salient subgoal.

3) Updating the Salient Subgoal Sets: The salient subgoal set $\mathcal{G}^{sub}_{SDM}(\bar{G})$ corresponding to *each* quantized proxy goal $\bar{G}$ in the *Saliency Table* is updated as follows:

$$\mathcal{G}^{sub}_{SDM}(\bar{G}) = \begin{cases} \mathcal{G}^{sub}_{SDM}(\bar{G}) + \{\bar{g}\}, & \textbf{if } \psi(\bar{g},\bar{G}) = 1 \\ & \quad \wedge\ \bar{s} \notin \mathcal{G}^{sub}_{SDM}(\bar{G}) \\ \mathcal{G}^{sub}_{SDM}(\bar{G}) - \{\bar{g}\}, & \textbf{if } \psi(\bar{g},\bar{G}) = 0 \\ & \quad \wedge\ \bar{s} \in \mathcal{G}^{sub}_{SDM}(\bar{G}) // \textit{pruning} \end{cases} \tag{6}$$

Equation 6 implies that a quantized subgoal $\bar{g}$ is added to the subgoal set $\mathcal{G}^{sub}_{SDM}(\bar{G})$ if it is salient ($\psi(\bar{g},\bar{G}) = 1$) and not already present in the set. On the other hand, $\bar{g}$ is removed from $\mathcal{G}^{sub}_{SDM}(\bar{G})$ if it is present in the set but not salient anymore. This is called pruning. This update operation for each $\bar{G}$ in the *Saliency Table* is performed after every SRR^{th} episode.

As mentioned in subsection IV-A, the LIDOSS agent gets the salient subgoal set corresponding to an exogenous goal at the beginning of every episode and uses that as the output space of the highest-level policy. If the non-quantized exogenous goal is G, the salient subgoal set is obtained as $\mathcal{G}^{sub}_{SDM}(G)$ = $\mathcal{G}^{sub}_{SDM}(\bar{G})$, if $\bar{G} = \lfloor \frac{G}{q^G} \rfloor \times q^G$, as per equation 4.

4) Remarks on the Subgoal Discovery Heuristic: The subgoal discovery heuristic, described above, does not use the rewards from the core UMDP. Hence, it is difficult to provide a theoretical analysis of the heuristic from the policy optimality perspective. The heuristic is designed to work without reward-based supervision and to find salient subgoal states that include topological bottlenecks in the state space. Hence, the subgoal discovery uses the occurrence probability of the subgoals to identify the salient subgoals. The efficacy of this heuristic is *empirically* shown in the task domains which contain bottlenecks (refer to the experiments discussed in section V). Hence, LIDOSS is more suitable than a simple *uniform* sampling of subgoals if the task domain contains bottlenecks.

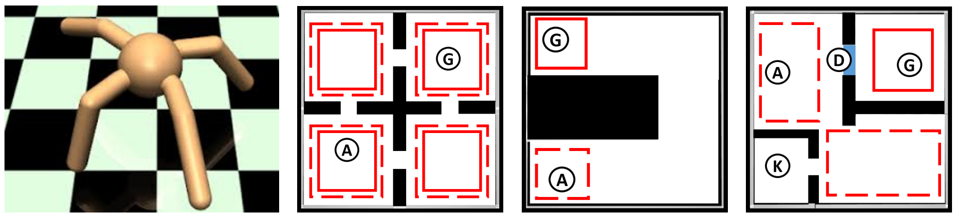
Moreover, subgoal discovery occurs continuously as long as new data is made available through the exploration performed by the agent. Hence, the set of outputs of the highest-level policy of LIDOSS keeps changing dynamically. This did not pose a practical issue in the task domains used in our experiments (section V), in which the subgoal space is bounded and not very large, since new subgoals are not discovered after the agent has completely explored the subgoal space. However, learning may not be stable in other task domains with a very large or infinite subgoal space.

Algorithm 1 Subgoal Discovery Procedure of LIDOSS

```
Initialize $q$, $q^G$, $SRR$, $|LMX|$, $\psi_{th}$ (refer to Table I)
Initialize $n_{table} \leftarrow 0$                    ▷ Size of Saliency Table
Initialize $nTrain \leftarrow$ number of training episodes
foreach EP $\in [0, nTrain)$ do
    Run the training episode
    Save the state-transition trajectory $\Gamma_s$ at the end of the episode
    Initialize subgoal trajectory $\Gamma_g \leftarrow \emptyset$
    foreach $s \in \Gamma_s$ do
        Insert subgoal $\phi^{sub}(s)$ in $\Gamma_g$          ▷ Abstraction
    $s_T \leftarrow$ terminal state in $\Gamma_s$.
    Proxy goal G $\leftarrow \phi(s_T)$                   ▷ Abstraction
    Quantize proxy goal: $\bar{G} \leftarrow \lfloor \frac{G}{q^G} \rfloor \times q^G$     ▷ Quantization
    Initialize quantized subgoal trajectory $\Gamma'_{\bar{g}} \leftarrow \emptyset$
    prev $\leftarrow \emptyset$
    foreach $g \in \Gamma_g$ do
        Quantize subgoal: $\bar{g} \leftarrow \lfloor \frac{g}{q} \rfloor \times q$       ▷ Quantization
        if $\bar{g} \neq$ prev            ▷ avoid consecutive duplicates
        then
            Insert $\bar{g}$ in $\Gamma'_{\bar{g}}$
            prev $\leftarrow \bar{g}$
                  ▷ Time complexity of quantization: O($|\Gamma_g|$)
    foreach $\bar{g} \in \Gamma'_{\bar{g}}$ do
        if $(\bar{g}, \bar{G}) \notin$ Saliency Table then
            Insert $(\bar{g}, \bar{G})$ in Saliency Table.
            $n_{table} \leftarrow n_{table} + 1$
            Set $n(\bar{G}) \leftarrow 0$, $n(\bar{g}, \bar{G}) \leftarrow 0$, $\psi(\bar{g}, \bar{G}) \leftarrow 0$
        $n(\bar{G}) \leftarrow n(\bar{G}) + 1$                  ▷ Count increment
        $n(\bar{g}, \bar{G}) \leftarrow n(\bar{g}, \bar{G}) + 1$            ▷ Count increment
             ▷ Time complexity of count increments: O($|\Gamma'_{\bar{g}}|$)
    if (EP   mod   SRR) = 0 then
        foreach $(\bar{g}, \bar{G})$ in Saliency Table do
            Initialize $LMX(\bar{g}) \leftarrow \emptyset$
            Get $|LMX|$ nearest quantized subgoal neighbours of $\bar{g}$, add them to a temporary set $temp$
            foreach $\bar{g}' \in$ temp do
                if $(\bar{g}', \bar{G}) \in$ Saliency Table then
                    Insert $\bar{g}'$ into $LMX(\bar{g})$
                $p(\bar{g}|\bar{G}) = \frac{n(\bar{g},\bar{G})}{n(\bar{G})}$
                Determine saliency $\psi(\bar{g}, \bar{G})$ as per equation 5
                Update $\mathcal{G}^{sub}_{SDM}(\bar{G})$ as per equation 6
        ▷ Time complexity of the updates: O($n_{table} \times |LMX|$)
▷ Average time complexity per episode: $O(|\Gamma_g|)$ + $O(|\Gamma'_{\bar{g}}|)$ + $O(\frac{n_{table} \times |LMX|}{SRR})$
```

V. Experiments

This section describes the experiment setup to evaluate LIDOSS (subsection V-A), the methods it is compared with (subsection V-B), and the observed results along with their analysis (subsection V-C). A discussion about the computation time required by LIDOSS and HAC is provided in Appendix *D* of the Supplementary Material.



(a) Ant agent. (b) Ant Four Rooms. (c) Ant Maze. (d) Ant Key-Lock.

Fig. 3: MuJoCo Ant environments for continuous control tasks used in our experiments. The labels are as follows: 'A' is Agent, 'G' is exogenous Goal, 'K' is Key, and 'D' is Door. The solid-border boxes show the regions within which a goal can be specified. The dashed-border boxes show the regions within which the agent can start.

A. *Setup*

The experiments are performed using a set of *simulated* MuJoCo continuous control task domains [9], described later. The task environments are deterministic. The tasks require goal-directed navigation, in which the agent has to traverse through the continuous state space using continuous primitive actions, to reach the target goal state(s). The task environment The agent is a simulated Ant robot with a torso and four limbs, depicted in Fig. 3(a). The state, goal, subgoal, and primitive action spaces are as follows:

The state space $\mathbb{S}$ is a continuous vector space with 29 dimensions. The first 3 dimensions represent the location of the Ant agent, denoted as (X, Y, Z). The 16^{th} and 17^{th} dimensions represent the velocity of the Ant in the directions of X and Y, respectively. The rest of the dimensions represent the orientation of Ant's torso and joints, and the velocities of the joints. The primitive action space $\mathbb{A}$ is eight-dimensional and continuous. Various dimensions of the action space represent the torques to be applied on the Ant's limb joints.

The goal space $\mathbb{G}$ is three-dimensional and continuous. It is obtained using the predefined abstraction function $\phi(s)$ (refer to subsection III-B), which extracts the first three dimensions from a state $s \in \mathbb{S}$ as a goal $G \in \mathbb{G}$. Therefore, a goal is a target location in the three-dimensional navigation environment.

The subgoal space $\mathbb{G}^{sub}$ is five-dimensional and continuous. The predefined subgoal abstraction function $\phi^{sub}(s)$ (refer to subsection III-B) extracts the first three dimensions (location (X, Y, Z)) and the 16^{th} and 17^{th} dimensions (velocity) from a state $s \in \mathbb{S}$ as a subgoal $g \in \mathbb{G}^{sub}$. Thus, a subgoal is an *intermediate* location and the velocity at that location that the agent should achieve to potentially reach the goal location.

All the spaces described above are taken exactly as defined by Levy et al. [2] for HAC, which is used as the baseline method in the experiments.

For all experiments, the training and testing is done in *alternating* batches of 100 episodes each, similar as the procedure used by Levy et al. [2] for HAC. One experiment trial consists of multiple alternating training and testing batches. For each testing batch, the percentage of the episodes (out of 100) in which the agent successfully reaches the *exogenous* goal is recorded as the *success rate* in that batch. The success rate is used as the performance measure during a trial. The experiments are divided into two types: *fixed goal experiments* and *dynamic goal experiments*. In the fixed goal experiments, the exogenous goal is randomly selected from the goal space at the beginning of the first episode of a trial. The same goal is then selected for the entirety of every subsequent episode in that trial. The purpose of this setup is to test how quickly does the agent learn to reach a particular goal from different *distant* start states. In the dynamic goal experiments, the exogenous goal is randomly selected at the beginning of each new episode in a single trial. The purpose of this setup is to test how quickly does the agent generalize to dynamically changing goals.

The MuJoCo task domains used for the experiments are shown in Fig. 3. The Ant Four Rooms domain [2] requires the agent (labelled 'A') to navigate from one room to another to reach the exogenous goal (labelled 'G'). The rooms are separated by walls and connected by passages (bottlenecks). We perform both fixed goal and dynamic goal experiments in this domain. The Ant Maze domain [3] requires the agent to navigate a '$\sqsupset$' shaped corridor to reach the exogenous goal. We perform only fixed goal experiments in this domain. The Ant Key-Lock is our custom task domain in which the agent has to fetch a key (bottleneck, labelled 'K') from one room before it can open a closed door (bottleneck, labelled 'D') and reach the exogenous goal in another room. We perform only fixed goal experiments in this domain. Additionally, we also perform dynamic goal and fixed goal experiments in a very simple task domain called Ant Reacher [2], which does not contain obstacles or bottlenecks. Further details about the task domains are provided in Appendix *B* of the Supplementary Material.

The hyper-parameters are the same across all tasks. The quantization sizes are $q = q^G = 0.5$ for the subgoal space and the goal space. The saliency threshold ψ_{th} is 0.3. The SDM refresh rate is $SRR = 50$. The number of neighbours in the LMX kernel are $|LMX| = 24$. The ϵ value of the highest-level policy of LIDOSS is equal to 0.3 during training and 0.0 during testing. Each episode is terminated after 700 time steps. The subgoal repetition time at the highest-level is $c^{HL} = 100$ time steps and at the intermediate-level it is $c^{HL-1} = 10$ time steps. The core UMDP reward (R) is -1 at each step and 0 only when the agent is in proximity to the exogenous goal. Similarly, the internal reward of the HRL agent (R_{in}; refer to subsection III-B) is -1 at each step and 0 only when the agent is in proximity to a subgoal. Please refer to Appendix *B* of the Supplementary Material for the details about how the

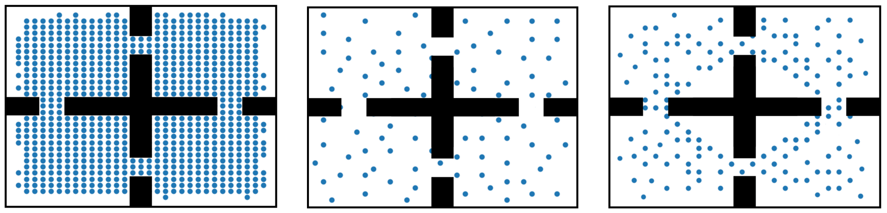

(a) HAC+Qn Subgoals. (b) HAC+QnFPS Subgoals. (c) LIDOSS Subgoals.

Fig. 4: Distribution of the subgoals discovered using different heuristics, in the Ant Four Rooms domain. This is a projection of the subgoals on a two-dimensional plane for the purpose of visualization.

proximity to an exogenous goal or a subgoal is determined.

As discussed in subsection IV-B, the subgoal discovery heuristic of LIDOSS sub-samples a discrete set of salient subgoals from the continuous subgoal space. The degree of discretization is controlled by the hyper-parameters q and $|LMX|$. The details about how we set these hyper-parameters are provided in Appendix *C* of the Supplementary Material.

B. Methods Compared

LIDOSS is compared against the following methods in the experiments:

Hierarchical Actor Critic (HAC) [2]: This method is used as the standard end-to-end Feudal HRL baseline without subgoal discovery or quantization. For details, refer to subsection III-C.

HAC + Quantization (HAC+Qn): This method is designed by ablating the subgoal discovery heuristic of LIDOSS and using the discrete set of subgoals, obtained only by the quantization of the continuous subgoal space (equation 3), as the outputs of the highest-level ϵ-greedy policy. For fair comparison, the quantization size is set as $q = 0.5$ similar to LIDOSS. Quantization provides a discrete set of subgoals distributed in a uniform grid-pattern over the subgoal space (Fig. 4(a)). HAC+Qn also works incrementally like LIDOSS, by periodically quantizing the subgoals which are the abstractions of the *explored* states, at an interval equal to the SDM Refresh Rate (SRR) of LIDOSS.

HAC + Quantization + FPS (HAC+QnFPS): This is an alternative method for salient subgoal selection. It is designed by replacing the probability-based local salient subgoal discovery heuristic of LIDOSS with the Farthest Point Sampling (FPS) algorithm [30]. The FPS algorithm starts by randomly sampling one of the quantized subgoals. Then, it samples another quantized subgoal which is the farthest from the first one, where the distance measure is the Euclidean distance in the subgoal space. Subsequently, it samples a third quantized subgoal which is the farthest from the first two, and so on. In this manner, it discovers 300 subgoals[5] to form the discrete set of outputs of the highest-level ϵ-greedy policy of HAC+QnFPS agent. The FPS algorithm does not depend on the exogenous or proxy goals, hence HAC+QnFPS discovers only one common set of subgoals. The subgoals are widely distributed over the subgoal space with similar concentration in different regions (Fig. 4(b)). Their arrangement is not directed towards bottlenecks, unlike the subgoal distribution of LIDOSS (Fig. 4(c)). HAC+QnFPS also works incrementally by applying the FPS sampling at an interval equal to the SDM Refresh Rate (SRR) of LIDOSS.

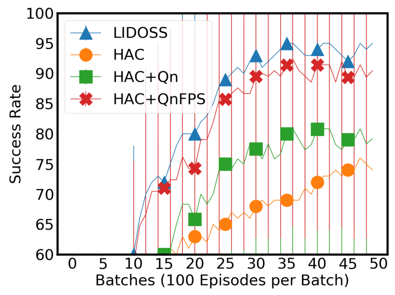
[5]LIDOSS discovers approximately 300 subgoals corresponding to a proxy goal in the different task domains by the end of training.

C. Results and Discussion

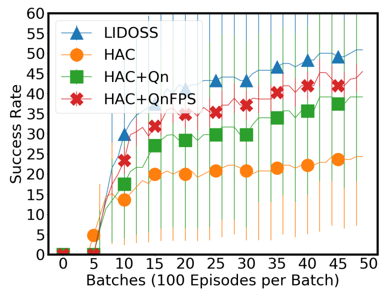
The results for the fixed goal and dynamic goal experiments performed in different task domains are shown in Fig. 5. As mentioned earlier, the training and testing batches are executed alternately, where each batch is of 100 episodes. Thus, the success rate after N testing batches also indicates the performance of an agent after N training batches. All methods perform comparably in the Ant Reacher domain (Fig. 5(e)), with similar results for both dynamic goal and fixed goal setup. This indicates that neither quantization nor subgoal discovery has a beneficial effect if the topology of the environment is very simple, that is, it contains no obstacle or bottleneck, and the agent can simply navigate straight to the goal in an open space. The rest of the domains have a more complicated topology, including obstacles and bottlenecks such as doors, passages, and key (Fig. 3). In those domains, various methods show a noticeable difference in performance.

Before analyzing the results further, the following points need to be recalled from section III: The efficacy of end-to-end HRL depends on both the *temporal abstraction* of the subgoals chosen by the highest-level policy and the *reachability* of those subgoals using the lowest-level policy. The highest-level policy learns to choose the subgoals with larger Q-values; such subgoals are farther from the current state and closer to the goal if the rewards are greater than or equal to zero only near the goal. This gradually leads to temporal abstraction during training, in combination with the repetition of a chosen subgoal for multiple time steps. At the same time, the highest-level policy also receives a large penalty for choosing a subgoal that cannot be reached using the lowest-level policy within limited time steps, which ensures the reachability of the subgoals.

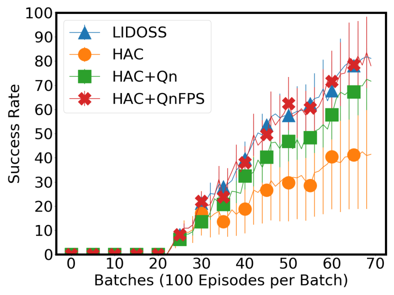
Now, the performance of various methods is analyzed as follows. LIDOSS, HAC+Qn, and HAC+QnFPS achieve better *final* success rate than HAC, to varying degrees. The common aspect of these three methods is the non-parametric ϵ-greedy policy at the highest level of the HRL agent's hierarchy of



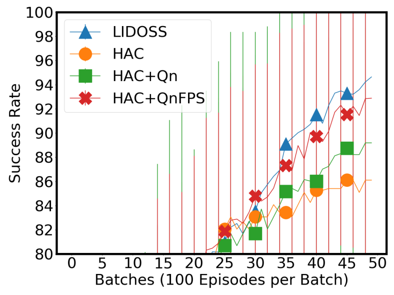
(a) Ant Four Rooms Fixed Goal.



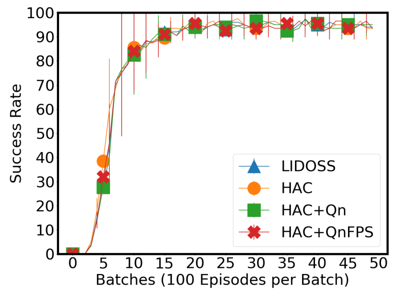
(b) Ant Key-Lock Fixed Goal.



(c) Ant Maze Fixed Goal.



(d) Ant Four Rooms Dynamic Goal.



(e) Ant Reacher Dynamic/Fixed Goal

Fig. 5: Success Rates in different task domains, calculated per 100 episodes in a testing batch. The result for each method is an average of 50 trials. The x-axis shows the number of finished testing batches, which is also equal to the number of finished training batches. The scale of the y-axis is restricted to show the differences between the success rates more clearly.

policies. As discussed in subsections III-C and IV-A, the parametric HAC policy incrementally shifts its output over the continuous subgoal space, based on the gradient of the Q-value. Thus, HAC requires more training time until the highest-level policy output converges near the region of the subgoal space with larger Q-values. On the other hand, the ϵ-greedy policies of LIDOSS, HAC+Qn, and HAC+QnFPS directly choose a subgoal from the discrete set of subgoals, based on the largest Q-value (since ϵ value is zero during testing). Hence, the performance benefit for LIDOSS, HAC+Qn, and HAC+QnFPS over HAC comes from avoiding the overhead of learning a parametric continuous-action policy at the highest level. The default learning rate for the parameters of HAC's highest-level policy is 0.001. We also tested HAC by decreasing and increasing the learning rate, but it still achieved lower success rates than other methods.

1) LIDOSS versus Pure Quantization: The results also show that quantization alone (that is, HAC+Qn) does not lead to the best gain in performance over HAC (Fig. 5). HAC+Qn uses a set of subgoals that are very densely distributed over the subgoal space (Fig. 4(a)). In contrast, LIDOSS discovers a set of subgoals that are sparsely distributed over the subgoal space (Fig. 4(c)). We observed that the sparse distribution provides an additional temporal abstraction effect to the LIDOSS agent. The highest-level policy of the HAC+Qn agent chooses a long and haphazard sequence of subgoals in a few episodes. This sometimes restricts the exploration done by the HAC+Qn agent, which might also affect learning. It happens due to the quantized subgoals being very close to each other. On the other hand, the highest-level policy of LIDOSS chooses subgoals that are naturally distant from each other, in terms of the number of time steps needed to reach them, because of the sparsity. This additional temporal abstraction effect helps in exploration and learning, leading to a better performance shown by LIDOSS.

2) LIDOSS versus other Subgoal Discovery Heuristics: LIDOSS performs slightly better than HAC+QnFPS in the task domains which contain bottlenecks, that are Ant Four Rooms and Ant Key Lock (please see Figure 5 in this document for the experiment results). This is because HAC+QnFPS does not explicitly discover the subgoals that lie at or near the bottlenecks in the Four Rooms and Key-Lock domains, since the FPS algorithm samples the subgoals that are more uniformly distributed over the subgoal space. In those domains, we observed that the HAC+QnFPS agent gets stuck at the walls in a few cases and it is not able to traverse through the bottlenecks, while the LIDOSS agent chooses the bottleneck subgoals and traverses through them. However, in the Ant Maze domain without bottlenecks, the two methods perform comparably. Hence, LIDOSS is more suitable for the task domains containing bottlenecks.

Finally, we also tested LIDOSS after ablating the Local Max kernel and only extracting the top 300 subgoals corresponding to a proxy goal, sorted by their occurrence probabilities compared globally. A majority of such subgoals are concentrated near the bottlenecks. The final performance of that agent is found to be similar to LIDOSS in the Four Rooms domain but worse than LIDOSS in the Key-Lock domain. The bottlenecks

Task	LIDOSS 3-levels	HAC 3-levels	LIDOSS 2-levels	HAC 2-levels	1-level
Four Rooms - fixed goal	94.8 ± 27.6	74.2 ± 29.5	36.3 ± 28.3	37.1 ± 30.2	6.7 ± 3.2
Four Rooms - dynamic goal	94.6 ± 28.3	85.9 ± 27.2	38.4 ± 31.1	44.7 ± 33.4	7.4 ± 3.5
Key Lock	50.6 ± 27.2	24.8 ± 18.1	8.5 ± 6.3	10.2 ± 7.1	1.1 ± 0.2
Maze	81.2 ± 14.4	41.6 ± 22.2	2.3 ± 0.4	5.5 ± 1.2	0.5 ± 0.2

TABLE II: The final success rate of an agent as a function of the depth of its hierarchy of policies, recorded after 70 testing batches in Ant Maze and 50 testing batches in other tasks. The 1-level agent is a standard RL agent with a single policy and no hierarchy or subgoals. The results for each task are the average of 50 trials.

in the Four Rooms domain (Fig. 3(b)) are the passages which can be easily reached from the adjacent rooms. However, the bottlenecks in the Key-Lock domain (Fig. 3(d)) are harder to reach from certain states. For e.g., the Key is harder to reach from outside the Key (K) room and the Door to the goal room (D) is harder to reach from the Key room, via a simple traversal on a straight path. In such cases, the reachability of the intermediate states also matters, which is achieved by the local selection of the salient subgoals using the Local Max kernel of LIDOSS (subsection IV-B).

3) Effect of the Depth of the HRL Agent's Hierarchy of Policies: As discussed in subsection III-B, a hierarchy of policies with more than two levels possibly offers the advantage of greater temporal abstraction at the highest level while also maintaining the reachability of the subgoals at the lowest level. On the other hand, in a two-level hierarchy, the highest-level policy has to choose the subgoals which are closer to the current state to avoid excess penalties if the agent cannot reach the subgoals (refer to *subgoal testing* in subsection III-C). This causes the highest-level policy to choose a longer sequence of subgoals to reach an exogenous goal within an episode, reducing the benefit from temporal abstraction and increasing the required training time. This trade-off between temporal abstraction and subgoal reachability is most likely the reason that a 3-level HRL agent performs better than a 2-level HRL agent for both HAC and LIDOSS, as reported in Table II. We also test a 1-level non-hierarchical agent, which is a standard RL policy that takes a state and a goal of the core UMDP as the inputs, and chooses a primitive action from the continuous action space as the output. This agent does not use any subgoal, hence it has to reach the long-horizon goal using only the primitive actions. It can be seen that the 1-level agent performs the worst in all the domains.

VI. Conclusion

This paper presents a method called Hierarchical Reinforcement **L**earning with **I**ntegrated **D**iscovery **O**f **S**alient **S**ubgoals (LIDOSS) which integrates a subgoal discovery heuristic into the end-to-end learning of an HRL agent's hierarchy of policies. The heuristic discovers a *sparse and discrete* set of salient subgoals from a large continuous subgoal space. The salient subgoals are *simultaneously* used as the output of the highest-level policy of the LIDOSS agent. The subgoal discovery heuristic is based on the local comparison of the probabilities of occurrence of different subgoals on various state-transition trajectories leading to the goal(s). It finds a set of subgoals that includes those lying at the bottleneck regions of the state space.

LIDOSS is compared with a recently introduced end-to-end HRL method called Hierarchical Actor Critic (HAC) [2] in a set of continuous control tasks in the MuJoCo task domains [9]. In HAC, the highest-level policy directly uses the large continuous subgoal space as its output space, without subgoal discovery. The experiments show that LIDOSS performs better than HAC in all task domains that contain obstacles or bottlenecks. The better performance of LIDOSS is mainly due to the use of a discrete-action highest-level policy and the reduced overhead of learning a continuous-action policy, such as in HAC.

a) Advantages and Limitations: LIDOSS has the following main ***advantages***: (i) It reduces the overhead of learning a continuous-action highest-level policy over a continuous subgoal space. (ii) It explicitly includes the subgoal states lying at the bottleneck regions, among other discovered subgoals, which is beneficial in the task domains containing the state-space bottlenecks. LIDOSS also has a few key ***limitations*** as follows: (i) LIDOSS might not have an advantage over other subgoal discovery heuristics, such as the Farthest Point Sampling (FPS) [30] used in our experiments, in task domains that do not contain bottlenecks. (ii) LIDOSS requires quantization of the subgoal space, which might be difficult if the subgoal space is high-dimensional, due to the curse of dimensionality. (iii) LIDOSS is not suitable for task domains with a very large (potentially infinite) state space because the *continuous* subgoal discovery makes the set of outputs of the highest-level policy unstable, which affects learning.

b) Future Work: To address the second limitation, we plan to replace the quantization operation with subgoal clustering using a learned subgoal similarity function in the future work. One scheme to learn such a similarity function is to label two subgoals as *similar* (label value equal to 1) if they are separated by a limited number of time steps on an episodic trajectory, otherwise, label them as *dissimilar* (label value equal to 0). A neural network can be pre-trained using this labelled data in a supervised manner to predict the similarity or dissimilarity of two high-dimensional subgoals. A similar scheme is used in a recent work on topological memory-based planning [31]. During the training of the HRL agent, the pre-trained similarity function can be used to find the similar subgoals and cluster them into a discrete centroid subgoal, which will provide the same effect as quantization.

LIDOSS is a model-free HRL method which requires a significant amount of data to learn the hierarchy of policies from scratch to reach new goals. In the future, we will investigate the task or goal decomposition from a different perspective, which is aligned with model-based planning [31],

[32]. The agent will use the exploration data to build state-transition models at different levels of *temporal abstraction*, it will use those models to sample the subgoals, and use planning to find the paths to the goal(s) through the subgoals. A model-based approach can help the agent memorize the structure or topology of the state space and subgoal space, and reduce the amount of data required to learn to reach new goals.

Acknowledgment

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).

References

[1] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete event dynamic systems*, vol. 13, no. 1-2, pp. 41–77, 2003.

[2] A. Levy, G. D. Konidaris, R. P. Jr., and K. Saenko, "Learning multi-level hierarchies with hindsight," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9*, 2019.

[3] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 3307–3317.

[4] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, 2017, p. 3540–3549.

[5] N. Dilokthanakul, C. Kaplanis, N. Pawlowski, and M. Shanahan, "Feature control as intrinsic motivation for hierarchical reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3409–3418, 2019.

[6] C. Sun, W. Liu, and L. Dong, "Reinforcement learning with task decomposition for cooperative multiagent systems," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2020.

[7] Z. Yang, K. Merrick, L. Jin, and H. A. Abbass, "Hierarchical deep reinforcement learning for continuous action control," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 11, pp. 5174–5184, 2018.

[8] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 5. Morgan-Kaufmann, 1993, pp. 271–278.

[9] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[10] S. Pateria, B. Subagdja, and A. H. Tan, "Hierarchical reinforcement learning with integrated discovery of salient subgoals," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '20. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2020, p. 1963–1965.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[12] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017, pp. 5048–5058.

[13] A. McGovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," in *Proceedings of the Eighteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 2001, pp. 361–368.

[14] O. Şimşek and A. G. Barto, "Skill characterization based on betweenness," in *Proceedings of the 21st International Conference on Neural Information Processing Systems*, ser. NIPS'08. Red Hook, NY, USA: Curran Associates Inc., 2008, p. 1497–1504.

[15] S. Mannor, I. Menache, A. Hoze, and U. Klein, "Dynamic abstraction in reinforcement learning via clustering," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 71.

[16] I. Menache, S. Mannor, and N. Shimkin, "Q-cut - dynamic discovery of sub-goals in reinforcement learning," in *Proceedings of the 13th European Conference on Machine Learning*, ser. ECML '02. Berlin, Heidelberg: Springer-Verlag, 2002, p. 295–306.

[17] O. Şimşek, A. P. Wolfe, and A. G. Barto, "Identifying useful subgoals in reinforcement learning by local graph partitioning," ser. ICML '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 816–823.

[18] M. C. Machado, M. G. Bellemare, and M. Bowling, "A laplacian framework for option discovery in reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 2295–2304.

[19] A. S. Lakshminarayanan, R. Krishnamurthy, P. Kumar, and B. Ravindran, "Option discovery in hierarchical reinforcement learning using spatio-temporal clustering." [Online]. Available: arXiv:1605.05359

[20] S. Zhou, Z. Xu, and F. Liu, "Method for determining the optimal number of clusters based on agglomerative hierarchical clustering," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 12, pp. 3007–3017, 2017.

[21] P. Yildirim and D. Birant, "K-linkage: A new agglomerative approach for hierarchical clustering," *Advances in Electrical and Computer Engineering*, vol. 17, no. 4, pp. 77–88, 2017.

[22] W. Bi and J. T. Kwok, "Mandatory leaf node prediction in hierarchical multilabel classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 12, pp. 2275–2287, 2014.

[23] J. Park and J. Kim, "Incremental class learning for hierarchical classification," *IEEE Transactions on Cybernetics*, vol. 50, no. 1, pp. 178–189, 2020.

[24] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1312–1320.

[25] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proceedings of Machine Learning Research*, vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1329–1338.

[26] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, no. 1–2, p. 181–211, Aug. 1999.

[27] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016*, 2016.

[28] P.-L. Bacon, *On the Bottleneck Concept for Options Discovery: Theoretical Underpinnings and Extension in Continuous State Spaces*, ser. Master's theses. McGill University Libraries, 2014. [Online]. Available: https://books.google.com.sg/books?id=40WSoAEACAAJ

[29] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 1479–1487.

[30] C. Moenning and N. A. Dodgson, "Fast marching farthest point sampling," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-562, Apr. 2003. [Online]. Available: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-562.pdf

[31] N. Savinov, A. Dosovitskiy, and V. Koltun, "Semi-parametric topological memory for navigation," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018*, 2018.

[32] B. Eysenbach, R. R. Salakhutdinov, and S. Levine, "Search on the replay buffer: Bridging planning and reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 15 246–15 257.