Neurocomputing xxx (xxxx) xxx

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward

Nan Jiang[a,b,c,d], Sheng Jin[a,b,c,d], Changshui Zhang[a,b,c,d,*]

[a] Institute for Artificial Intelligence,Tsinghua University (THUAI), Beijing, P.R. China
[b] State Key Lab of Intelligent Technologies and Systems, Beijing, P.R. China
[c] Beijing National Research Center for Information Science and Technology (BNRist), Beijing, P.R. China
[d] Department of Automation, Tsinghua University, Beijing, P.R. China

ARTICLE INFO

Article history:
Received 16 October 2018
Revised 11 March 2019
Accepted 3 June 2019
Available online xxx

Communicated by Dr. Kee-Eung Kim

Keywords:
Hierarchical reinforcement learning
Automatic curriculum learning
Sparse reward reinforcement learning
Sample-efficient reinforcement learning

ABSTRACT

Mastering the sparse reward or long-horizon task is critical but challenging in reinforcement learning. To tackle this problem, we propose a hierarchical automatic curriculum learning framework (HACL), which intrinsically motivates the agent to hierarchically and progressively explore environments. The agent is equipped with a target area during training. As the target area progressively grows, the agent learns to explore from near to far, in a curriculum fashion. The pseudo target-achieving reward converts the sparse reward into dense reward, thus the long-horizon difficulty is alleviated. The whole system makes hierarchical decisions, in which a high-level conductor travels through different targets, and a low-level executor operates in the original action space to complete the instructions given by the high-level conductor. Unlike many existing works that manually set curriculum training phases, in HACL, the total curriculum training process is automated and suits the agent's current exploration capability. Extensive experiments on three sparse reward tasks, long-horizon stochastic chain, grid maze, and the challenging Atari game Montezuma's Revenge, show that HACL achieves comparable or even better performance but with significantly less training frames.

1. Introduction

The navigation ability is a core element of artificial intelligence. It is a basic component of many intelligence-based applications, including but not limited to the household robot, unknown environment exploration, etc. Successful navigation requires an understanding of the environment, the current state, the final target, and the consequences of the actions. Recently deep reinforcement learning has shown fascinating achievements on many tasks, e.g. playing Atari games from raw pixels [1], solving complex board game Go [2], object detection [3,4], image captioning [5,6], and mastering locomotive control problems [7,8]. By utilizing deep neural networks to promote image understanding, deep reinforcement learning has been proposed to solve navigation tasks [9–13].

For the navigation task, it learns a policy that maximizes target place achieving reward. Rewards provide information about how good each state and action is. However, the sparse reward environments, which means non-zero rewards occur only at a tiny probability or after a long-horizon, pose a severe challenge. While many works on deep reinforcement learning achieve super-human performance on most Atari games, without informative exploration, they fail to learn a good policy on the infamous game Montezuma's revenge [1,7,14,15].

Recently, a few works have been proposed to tackle the sparse reward challenge. Some of them design prioritized experience replay [16] for more instructive learning, some utilize prior knowledge like demonstrations from experts [17–19], some employ the hierarchical reinforcement learning framework [20,21] or the intrinsic rewards [22–26] to guide exploration. These approaches can be used separately or in conjunction to alleviate the sparse reward difficulty. However, there exist some limitations of these works. By utilizing the demonstrations, it assumes the existence of successful experience, thus limiting their application in unknown or unfamiliar tasks. Some hierarchical reinforcement learning or intrinsic motivation approaches require the decomposable or reducible structure of the environment, such that downstream tasks could be employed to learn skills. With the help of the pre-defined hierarchy, some basic skills are trained in advance and reused in later complex tasks [27,28]. As for some other works which do not rely

* Corresponding author. Department of Automation, Tsinghua University, Beijing, P.R. China.
E-mail addresses: jiangn15@mails.tsinghua.edu.cn (N. Jiang), js17@mails.tsinghua.edu.cn (S. Jin), zcs@mail.tsinghua.edu.cn (C. Zhang).

https://doi.org/10.1016/j.neucom.2019.06.024
0925-2312/

on known hierarchy or demonstrations, they suffer from high demands of training steps [16,23].

We take the ideas of learning by steps, spatial-temporal abstraction, and knowledge reuse from human experiences aiming to increase the data efficiency of training in sparse reward navigation tasks. Three examples explain the ideas. When parents teach their children to walk, they put the candy or toys in front of the baby, and then move the toys further to encourage the baby to walk further and further. If toys had been put in the distance from the beginning, the baby wouldn't be able to walk a long way, but they would succeed by learning by steps. This idea is abstracted as curriculum learning [29] and has been applied to many works [10,30–34]. Also, human makes use of spatial-temporal abstraction to simplify the navigation process. To visit a friend in another city, we always plan the trip in three stages: travel from my home to the station, then take a train to that city, and at last from the train station to the friend's home. Hierarchical reinforcement learning employs this spatial-temporal abstraction. Hence it learns to make hierarchical decisions to operate on different time-scales. The last example shows how people reuse their knowledge when navigating. John walks to the subway station then takes the subway to work every day. The knowledge of the way to the subway can be reused when he needs to visit other places by subway. Such knowledge reuse is naive for human, but hard for the neural network based agent. So, we anticipate a knowledge reuse in our proposed framework.

To summarize, our framework is based on curriculum learning and hierarchical reinforcement learning. It contains three components: the high-level conductor, the low-level executor, and the state graph. The high-level conductor gives high-level commands to reach the target area or get extrinsic rewards. It operates at a coarse timescale. Meanwhile, its action is set as the subgoal of the low-level executor. The low-level executor, acting at a finer timescale, aims to reach its subgoal. Also, we take advantages of a state graph to enable automatic curriculum learning. While most existing works manually set the curriculum training phase, in our work the curriculum tasks are automatically generated during training and well adjusted to the agent's current navigation capability. Meanwhile, a novel sub-module called HC-explore for extensive exploration is introduced. The extensive exploration endows the agent with the ability to do macro path planning. And the knowledge reuse of the macro path planning further benefits the agent when navigating to long-horizon places. Benefiting from the ideas from human experiences, our hierarchical automatic curriculum learning framework (HACL) makes challenging sparse reward navigation task densely rewarded instead. Thus much fewer training frames are required to learn a good navigation policy.

HACL has the following advantages, and we give a detailed analysis of them in experiments.

1. **Spatial-temporal abstraction** effectively reduces the size of state space, allowing the automatically generated hierarchical structure to enable clever exploration.
2. **Curriculum setting** converts the environment to be dense-rewarded, and gradually but quickly guides the agent to explore further.
3. **Introduction of HC-explore** enables efficient knowledge reuse for long-horizon exploration by endowing the agent with the ability of macro-state path planning.

Combining the above three advantages, the HACL agent learns efficiently on the sparse reward tasks. Experiments show that our method dramatically reduces the training steps.

2. Notation and formulation

Reinforcement learning is based on the formulation of Markov Decision Process(MDP). A Markov Decision Process is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma >$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ defines the transition model, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ represents the reward function, and γ is the discount factor.

The agent's behavior, is defined by a policy, $\pi_\theta : \mathcal{S} \times \mathcal{A} \to [0, 1]$, which maps the states to a probability distribution over the actions. At each time step $t = 1, \ldots, T$, the agent stays at the state $s_t \in \mathcal{S}$ and selects action $a_t \in \mathcal{A}$ according to its policy π_θ to interact with the environment. The agent then receives a reward r_t and transits to the next state s_{t+1}, following the transition model $\mathcal{T}$. Reinforcement learning aims to learn a policy which maximizes the expected discounted future reward from the initial state, defined as

$$\begin{aligned} J_\pi &= E_{a_i \sim \pi}[R_{s_1}] \\ &= E_{a_i \sim \pi}[r_1 + \gamma r_2 + \ldots] \\ &= E_{a_i \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}\right] \end{aligned} \tag{1}$$

2.1. Hierarchical formulation

Hierarchical reinforcement learning, following the divide-and-conquer paradigm, decomposes a complex problem into some interrelated sub-problems. The options framework [35] is a general formulation, which temporally abstracts a sequence of primitive actions as an option. In addition to this temporal abstraction of actions, human tend to make use of their knowledge of the environment at multiple scales of spatial abstractions. e.g., as introduced in Section 1 we usually abstract a long journey as transitions between several landmarks. To summarize, we utilize spatial-temporal abstractions to make the agent explore efficiently.

We define a spatial abstraction of states as a macro-state M_i to dramatically reduce the state space. The macro-state space is denoted as $\mathcal{M}_t = \{M_i : i = 1, \ldots, N_t\}$, in which N_t is the number of macro-states at timestep t. More macro-states will be added into $\mathcal{M}_t$ while the agent travels further in the environment. Thus the size of macro-state space grows over time. Without ambiguity, we may ignore the subscript t to simplify notations in the following paragraphs.

The macro-state abstracts close states into a group. As some existing works manually define the hierarchy in advance [21], in HACL, each state in the environment is assigned to one macro-state based on a domain-specific distance measurement $Dist : \mathcal{S} \times \mathcal{M} \to \mathbb{R}$. We denote the macro-state, to which the state s_i is assigned, as $s_i^{(m)}$. Based on this, we introduce the *Coverage* notion of each macro-state:

Definition 2.1 (Coverage of macro-state)**.** For $\forall M_i \in \mathcal{M}$, its coverage is the set of states assigned to it: $C_i := \{s_j : s_j^{(m)} = M_i\}$

Macro-actions are the temporal abstractions of variable-length multistep primitive actions. We expect the macro-actions to perceive some meaningful action patterns. A macro-action is denoted with u. The agent executes a sequence of macro-actions $\{u_1, u_2, \ldots, u_m, \ldots\}$, and each macro-action represents a sequence of primitive actions, $u_m = \{a_1^{(m)}, \ldots, a_{n_m}^{(m)}\}$, in which n_m is the number of the primitive actions in macro-action u_m.

2.2. Curriculum formulation

Curriculum learning [29] makes the agent gradually learn to master a challenging task by constructing curricula of easy-to-hard

Please cite this article as: N. Jiang, S. Jin and C. Zhang, Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward, Neurocomputing, https://doi.org/10.1016/j.neucom.2019.06.024

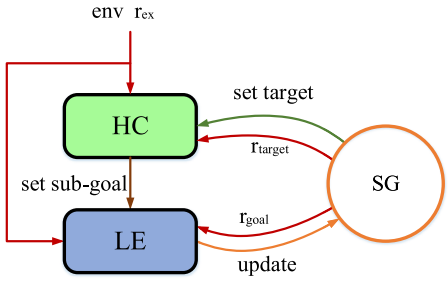


Fig. 1. Framework overview.

or clear-to-ambiguous tasks. Curriculum learning has been employed in many works [10,30,31]. However, most of these works make use of handcrafted curricula.

Therefore, we consider to build the curricula automatically: the training process automatically generates a sequence of tasks in a curriculum fashion. We first define the difficulty measurements, which helps define the notion of curriculum sequence.

Definition 2.2 (Difficulty function on state space)**.** We define the difficulty function on the state space $\mathcal{S}$. $\forall s_i \in \mathcal{S}$, difficulty function $f_D : \mathcal{S} \to \mathbb{R}$, maps the state to a difficulty level. $\forall s_i, s_j \in \mathcal{S}$, if $f_D(s_i) < f_D(s_j)$, we say state s_j is more difficult than state s_i.

The difficulty function measures how difficult a state is to reach. Induced by this, we could extend the difficulty metric on the macro-state.

Definition 2.3 (Difficulty function on macro-state space)**.** The difficulty function on the macro-state $\hat{f}_D : \mathcal{M} \to \mathbb{R}$, is induced by the difficulty function f_D, mapping the macro-state to a difficulty level.

$$\forall M_i \in \mathcal{M}, \hat{f}_D(M_i) := \sup_{s_j \in C_i} f_D(s_j)$$

in which C_i is the coverage of macro-state M_i.

Similarly, $\forall M_i, M_j \in \mathcal{M}$, if $\hat{f}_D(M_i) < \hat{f}_D(M_j)$, we say macro-state M_j is more difficult than M_i. Considering the definition of $\hat{f}_D$, we do not require all of the states in the coverage of macro-state M_j to be more difficult than those of macro-state M_i.

After introducing the notions of difficulty measurements, we give the curriculum definition in our framework. In this case, the HACL builds a curriculum sequence of macro-state sets as the agent explores further in the environment,. The definition of *Curriculum Sequence* is as follows.

Definition 2.4 (Curriculum Sequence of macro-state sets)**.** For an ordered sequence of macro-state sets $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n\}$, it is a curriculum sequence under difficulty function f_D if it satisfies: $\forall i = \{1, 2, \ldots, n-1\}$

$$sup_{M_k \in \mathcal{M}_i} \hat{f}_D(M_k) <= sup_{M_l \in \mathcal{M}_{i+1}} \hat{f}_D(M_l)$$

In the following, we will give the difficulty function f_D, and show that HACL builds a curriculum sequence of macro-state sets $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_t, \ldots\}$ under f_D during training.

3. Methods and models

3.1. Framework

The framework of our hierarchical automatic curriculum learning (HACL) is presented in Fig. 1. Three major components: the state graph (SG), the high-level conductor (HC) and the low-level executor (LE), work cooperatively in HACL.

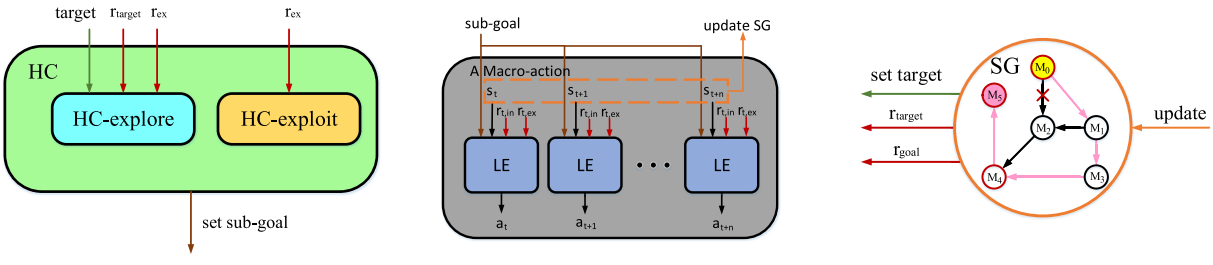
In HACL, the agent is equipped with a target set of macro-states. A curriculum sequence of target sets enables curriculum learning. By expanding the area of target sets, the agent learns to walk further. In order to achieve more extrinsic rewards or reaching the target macro-state M_{target} (sampled from the target set), the high-level conductor gives macro path planning on macro-states. The high-level conductor gives action of the next macro-state to reach, which is set as the subgoal g_T for the low-level executor. The low-level executor functions as an end-effector. It executes a sequence of primitive actions $\{a_1, a_2, \ldots, a_{n_T}\}$, in order to reach g_T. The trajectories $\{s_1, s_2, \ldots, s_{n_T}\}$ collected by the low-level executor are used to build and update the state graph. It models a coarse graph of transitions between macro-states. This graph provides hierarchy and curriculum information about the environment. Thus, target sets are generated based on the state graph. Also, it gives pseudo target-achieving rewards and goal-reaching rewards for the high-level conductor and low-level executor respectively. The target sets expand from near to far during learning and are well-adjusted to the agents current navigation capability. While extrinsic rewards are sparse, the pseudo target-achieving rewards and goal-reaching rewards are dense.

We would explain these three components in detail in the following sections.

3.2. High-level conductor

The High-level Conductor (HC) gives macro-state path planning, which serves as long-range subgoals for the low-level executor. By making use of hierarchical actions, it breaks the original long-horizon action sequences into several transitions between macro-states.

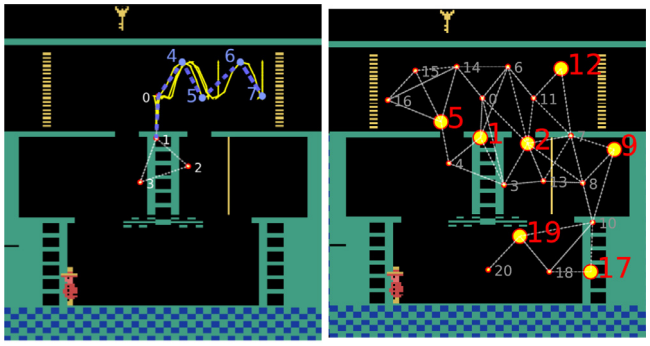
The high-level conductor contains two sub-modules, one for exploration (HC-explore), and the other for exploitation (HC-exploit). Both two high-level conductors learn a macro-state policy $\pi_H^{\theta_H}$: given the current macro-state M_T, with or without the target



(a) High-level Conductor (HC) (b) Low-level Executor (LE) (c) State Graph (SG)

Fig. 2. Three components of HACL.

Please cite this article as: N. Jiang, S. Jin and C. Zhang, Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward, Neurocomputing, https://doi.org/10.1016/j.neucom.2019.06.024



(a) Update of the state graph. (b) Expansion of the target set

Fig. 3. The existing state graph is marked with small white indexes and white transitions. Fig. 3a plots the agent's trajectory in yellow and four updated macro-states and their corresponding transitions in blue. The target set $\mathcal{T}$ in Fig. 3b is marked with large red indexes and yellow dots. $\mathcal{T}$ generates a curriculum sequence on the order of macro-state indexes {1, 2, 5, 9, 12, 17}.

macro-state M_{target}, it estimates a distribution of the next macro-states M_{T+1} to reach. The function of the high-level conductor is illustrated in Fig. 2a.

Two high-level conductors are trained in parallel with different rewards, thus with different training purposes. HC-explore aims to reach a high-level target macro-state M_{target}, which is sampled from the state graph, explicitly, its policy is $\pi_{H_{explore}}(M_{T+1}|M_T, M_{target}; \theta_{H_{explore}})$. HC-exploit, whose purpose is to maximize the extrinsic rewards, learns a policy $\pi_{H_{exploit}}(M_{T+1}|M_T; \theta_{H_{exploit}})$. Meanwhile, the predicted next macro-state M_{T+1} contemporarily acts as the subgoal g_{T+1} for the low-level executor.

HC-explore learns to do macro-state path planning on macro-states to reach a target macro-state M_{target}. As the target extends further in the environment, the agent could evoke its knowledge about nearby states to travel to further states and explore rarely visited areas quickly.

Each time the low-level executor terminates, the HC-explore receives its reward as follows.

$$r_{HC-explore} = c_1 \cdot \mathbb{1}_{\{target\ \ reached\}} + R_{ex} \tag{2}$$

The first term is a dense pseudo target-achieving reward on whether the agent has achieved M_{target}. It intrinsically rewards the HC-explore for traveling to target areas. The coefficient $c_1 (c_1 > 0)$ controls the strength of this item. The second term is the extrinsic rewards accumulated within the macro-action, i.e., the sequence of primitive actions executed by the low-level executor. It acts to remind the agent to get higher extrinsic rewards as well, which accords with the ultimate objective and HC-exploit's rewards.

HC-exploit aims to maximize the future extrinsic rewards. Therefore, eliminating the first item in Eq. (2), the HC-exploit is trained with sole external rewards. By sharing the low-level executor with HC-explore, HC-exploit benefits from HC-explore's macro-state path planning capability.

We use A3C [36] to train both high-level conductors, and add an entropy regularization term $H(\pi_H^{\theta_H})$ to boost exploration. The regularized expected reward is:

$$J(\pi_H^{\theta_H}|\pi_L^{\theta_L}) = \mathbb{E}_{\pi_H^{\theta_H}|\pi_L^{\theta_L}}\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}\right] + \beta H(\pi_H^{\theta_H}|\pi_L^{\theta_L}), \tag{3}$$

where $\pi_L^{\theta_L}$ represents the policy of the low-level executor. The strength of the entropy term is controlled by the hyper-parameter β.

The parameters of the actor-critic algorithm contain two parts, $\theta_H = \{\theta_H^\mu, \theta_H^v\}$, θ_H^μ for actor and θ_H^v for critic.

The gradient of θ_H^μ and θ_H^v is

$$\nabla_{\theta_H^\mu} J(\pi_H|\pi_L) = \nabla_{\theta_H^\mu} \log \pi\left(a_t|s_t;\theta_H^\mu\right)\left(R_H - V(s;\theta_H^v)\right) + \beta \nabla_{\theta_H^\mu} H\left(\pi\left(s_t;\theta_H^\mu\right)\right) \tag{4}$$

$$\nabla_{\theta_H^v} J(\pi_H|\pi_L) = \frac{\partial\left(R_H - V(s;\theta_H^v)\right)^2}{\partial\theta_H^v}, \tag{5}$$

where $V(s;\theta_H^v)$ is the critic function, acting as a baseline to reduce the variance of policy gradients, and R_H is the discounted reward. Note that the step rewards are different for two high-level conductors, as is introduced before.

3.3. Low-level executor

The low-level executor (LE) learns to achieve the low-level subgoal g_T, i.e., the action M_T given by the high-level conductor, as is shown in Fig. 2b. It operates at a finer timescale and directly interacts with the environment. At each time step, it selects a primitive action a_t based on both g_T and s_t. The low-level executor terminates when it reaches g_T or exceeds a certain step limit. Then, it is time for the high-level conductor to give the next macro-state and reset it as the new g_{T+1}.

The low-level executor is trained with a combination of intrinsic and extrinsic reward, see Eq. (6). The coefficient α $(0 < \alpha <= 1)$ controls the weight between them. The extrinsic reward is clipped to the same scale (controlled by c_2) with the intrinsic reward, which contains a goal-reaching reward and a punishment of equal value if the number of low-level steps exceeds a certain limit. Also the agent receives a tiny punishment each step to reach g_T within fewer timesteps ($c_2 > 0$, $c_3 \geq 0$):

$$r_{LE} = \alpha * [c_2 \cdot \mathbb{1}_{\{subgoal\ \ reached\}} - c_2 \cdot \mathbb{1}_{\{exceed\ \ step\ \ limit\}} - c_3] + (1-\alpha)\cdot \min(\max(r_{ex}, -c_2), c_2) \tag{6}$$

We use A3C and add an entropy regularization to train the low-level executor as well. The regularized expected reward and parameter gradients are similar to Eqs. (3)–(5) but with different rewards.

3.4. State graph

The state graph maintains the macro-states and transitions between them. HACL constructs the state graph from the the state sequence. Based on the state graph, a curriculum sequence of target set is generated during learning.

3.4.1. The update of the state graph

The state graph is represented as a dynamic undirectional graph $\mathcal{G}_t = \{\mathcal{M}_t, \mathcal{E}_t\}$, in which the vertices $\mathcal{M}_t = \{M_i : i = 1...N_t\}$ represent macro-states at time t and the edges $\mathcal{E}_t = \{e_{ij} \in \{0,1\} : i = 1...N_t, j = 1...N_t, i \neq j\}$ represent transitions between macro-states. If $e_{ij} = 1$, there exists a transition between macro-state M_i and M_j. N_t is the number of macro-states in the state graph at time t. As the agent travels to distant places, more macro-states will be added into $\mathcal{M}_t$. Without ambiguity, we may ignore the subscript t to simplify notations in the following paragraphs.

The state graph $\mathcal{G}$ is updated by the agent's trajectories. The expansion of $\mathcal{G}_t$ relates to the growth of agent's exploration radius of the environment. As stated in Section 2.1, each state in the environment could be assigned to one macro-state based on a distance measurement *Dist*. An accurate distance function requires domain-specific knowledge. However, in this work, for the sake of simplicity, each state is represented with the agent's position and the euclidean distance applies in all the tasks.

Please cite this article as: N. Jiang, S. Jin and C. Zhang, Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward, Neurocomputing, https://doi.org/10.1016/j.neucom.2019.06.024

But there does exist a hyper-parameter which requires expertise knowledge and is crucial for learning, the threshold distance δ for adding a new macro-state. If the agent discovers a state s_i, with which $\min_{j=1...N_t} Dist(s_i, M_j) > \delta$, a new macro-state is added into the macro-state space $\mathcal{M}_t$. Otherwise, this state is assigned to the macro-state the most similar to it, denoted as $s_i^{(m)} = \arg\min_{j=1...N_t} Dist(s_i, M_j)$. Then the trajectory of states could be converted to transitions between macro-states, and corresponding edges will be updated in $\mathcal{E}_t$. Algorithm 1 lists the process of updating $\mathcal{G}$ with trajectory $s_{\{0,...,n\}}$, and Fig. 3a shows an example update. The agent's trajectory is plotted with yellow line. Because of the dead transitions in the last part of the trajectory, a fixed length at the end of the trajectory is clipped. The existing state graph contains 4 macro-states, indexed from 0 to 3. Transitions between them are illustrated with white lines. Four new macro-states and their corresponding transitions are updated in $\mathcal{G}$, as the blue dots and lines shows.

Algorithm 1 UpdateG($s_{\{0..n\}}$, $\mathcal{G}_t$, l, δ).

Input: state sequence $s_0, s_1, \ldots, s_n$, current state graph $\mathcal{G}_t = \{\mathcal{M}_t, \mathcal{E}_t\}, N_t = \|\mathcal{M}_t\|$; clip length l, distance threshold δ.

for $i = 0$ to $n - l$ **do**
  if $\min_{j=1,2,...,N_t} dist(s_i, M_j) > \delta$ **then**
    add M_{N_t} into $\mathcal{G}$, M_{N_t} locates at s_i's coordinate
    $N_t = N_t + 1$
  end if
end for
$k = \arg\min_{j=1,2,...,N_t} dist(s_0, M_j)$
for $i = 1$ to $n - l$ **do**
  $j = \arg\min_{j=1,...,N_t} dist(s_i, M_j)$
  $e_{kj} = e_{jk} = 1$
  $k = j$
end for

3.4.2. The expansion of the target set

The state graph contains connectivity information between macro-states, therefore is critical for the curriculum construction of target sets. However, the macro-state space $\mathcal{M}_t$ itself is not suitable for directly functioning as the target set.

Fig. 3b shows a state graph generated by 200 episodes of the agent's random trajectory. The agent explored some faraway states, like macro-state 17, 18, 19, 20, just by a random walk. These distant macro-states will be kept in $\mathcal{M}_t$ but are hard for a cold-started agent to reach repeatedly. Also, it is clear from Fig. 3b that the amount of the macro-states is large. It is impractical to make all the macro-states as the target macro-states, otherwise, the agent will improve little from each stage of curricula and the large amount of target set distracts the agent from each task's original purpose.

Considering the above two aspects, we sample a growing target set from macro-sate space $\mathcal{M}$ once a while. The action space of high-level conductors is limited to the macro-states in the target set as well. At each time, the target set suits the agent's current navigation capability. Consecutive stages of the target set forms a curriculum sequence.

To be specific, a high-level target set $\mathcal{T}_t = \{M_{t_i} : t_i \in \{1, \ldots, N_t\}, i = 1, \ldots, n_t\}$ is maintained, where n_t is the number of macro-states in the target set at time t. We define the difficulty function $f_D : \mathcal{S} \to \mathbb{R}$ as the expected steps the agent takes to travel from the initial state to state s. Thus, $\hat{f}_D(M_i)$ represents the expected number of steps from the initial state to M_i. If macro-states are selected in the order of this difficulty function, the sequence of target sets $\{\mathcal{T}_t\}_{t=1,2,3,...,}$ forms a curriculum sequence under f_D.

However, there leaves a question: how does HACL estimate $f_D(s)$ and $\hat{f}_D(M_i)$? A strikingly simple way is to select macro-states in order of their indexes. In fact, the exact value of $f_D(s)$ and $\hat{f}_D(M_i)$ is neither required nor estimated, but the order of $\hat{f}_D(M_i)|_{i=1,...,N_t}$ matters, as the definition of curriculum sequence is defined by the order of the difficulty level of the macro-states. The good news is that, the macro-states are added into $\mathcal{G}$ with an index in order with their trajectories. In other words, for any macro-state with index i, the macro-states on the sure way from initial state to it will have an index smaller than i.

Algorithm 2 ExpandT(K; $\mathcal{G}_t$, $\mathcal{T}_t$).

Input: current state graph $\mathcal{G}_t = \{\mathcal{M}_t, \mathcal{E}_t\}, N_t = \|\mathcal{M}_t\|$, current target set $\mathcal{T}_t$, $n_t = \|\mathcal{T}_t\|$, adding count K.

CoveredSet $C = \emptyset$
for M_{t_i} in $\mathcal{T}_t$ **do**
  add M_{t_i} and its neighbours into C
end for
for $k = 1$ to K **do**
  for $i = 1$ to N_t **do**
    if $i \notin C$ **then**
      add M_i into target set $\mathcal{T}_t$
      $n_t = n_t + 1$
      add M_i and its neighbours into C
      break
    end if
  end for
end for

During training, a high-level target M_{target} is randomly selected from last n macro-states in $\mathcal{T}_t$ for HC-explore. When the agent reaches the following condition: successfully achieving these target macro-states with a probability(p) or making high-level decisions for a sufficient number(D) of times, K new macro-states with higher difficulty level will be sampled from $\mathcal{M}_t$ and added into $\mathcal{T}_t$.

Algorithm 2 demonstrates the expansion of the target set. When the agent learns to achieve the target macro-states in the current target set, we assume all the neighbouring macro-states of these target macro-states are easily covered by the agent's navigation capability. So all the existing macro-states and their neighbours are marked as covered, and from the other macro-states in the state graph, HACL selects the one with the smallest index to be added into the target set. Fig. 2 shows an example $\mathcal{G}$ and $\mathcal{T}$. If starting from two macro-states in $\mathcal{T}_0$, and two new macro-states are added into $\mathcal{T}$ when expanding it, the generated curriculum sequence of $\mathcal{T}$ will be {1, 2}, {1, 2, 5, 9}, {1, 2, 5, 9, 12, 17}, {1, 2, 5, 9, 12, 17, 19}.

We must emphasize that, we only use the state graph to (1) generate the curriculum sequence $\{\mathcal{T}_t\}_{t=1,2,3,...,}$ (2) make use of the saved location of macro-states to give the intrinsic target-achieving and goal-reaching rewards. An existing work makes uses of A* algorithm to do high-level path planning [37]. HACL does not use the state graph as a map, but use the high-level conductor to learn the macro-state plan. Because of the unavoidable imperfection of distance function, $\mathcal{G}$ could be noisy and contains false transitions of macro-states. Fig. 2c shows an example state-graph with a misleading macro-state transition. The agent needs to travel from macro-state M_0 to the target M_5. The graph would give the shortest path: $M_0 \to M_2 \to M_4 \to M_5$, which is infeasible due to the misleading transition from M_0 to M_2. However, if the high-level conductor learns through trial and error, from the previous target-achieving reward, it would learn to give another feasible path through macro-state M_1 and M_3.

The whole learning process is tabulated in Algorithm 3, with HC-explore as the high-level conductor. As for the HC-exploit, it

Please cite this article as: N. Jiang, S. Jin and C. Zhang, Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward, Neurocomputing, https://doi.org/10.1016/j.neucom.2019.06.024

Algorithm 3 Training process of HACL (use HC-explore as high-level conductor).

Input: threshold δ, clip length l, start number of macro-states m_1, add number of macro-states K, low-level step limit s

build initial $\mathcal{G}$ from random trajectories
$\mathcal{T} = \emptyset$
ExpandT(m_1; $\mathcal{G}, \mathcal{T}$)
i = 0
while True **do**
  sample M_{target} from last n macro-states in $\mathcal{T}$
  while M_{target} not achieved and not terminate **do**
    HC-explore gives M_T
    set low-level subgoal $g_T = M_T$
    $R_{ex} = 0$
    $t = 0$
    the agents stays at s_0
    while g_T not achieved and $t < s$ and not terminated **do**
      LE takes an action
      $t = t + 1$
      the agent moves to new state s_t, and receives r_{ex}
      $R_{ex} = R_{ex} + r_{ex}$
      Build($s_{\{0..t\}}$; $\mathcal{G}, l, \delta$)
    end while
    train LE with the reward in Eq. (6)
    $i = i + 1$
  end while
  train the HC-explore with reward in Eq. (2)
  if terminated and reach the condition of expanding $\mathcal{T}$ **then**
    ExpandT(K; $\mathcal{G}, \mathcal{T}$)
  end if
end while

just ignores the function of M_{target} and removes the pseudo target-achieving reward. In the testing phase, only HC-exploit will be kept and guide the agent to get higher external rewards in the environment.

4. Experiments

In this section, we evaluate HACL on three difficult sparse reward environments: A long-horizon chain, grid maze, and Atari game Montezuma's Revenge.

In the first environment, the state graph is clean and perfect. Because of the walls in maze, there exist many false transitions in the state graph of this environment. The last one, Atari game Montezuma's Revenge, is a notoriously challenging sparse reward game. Many existing reinforcement learning algorithms fail to learn a meaningful policy [1,7,14,15], and even for those who achieve good results, either require expert demonstrations or a huge demand for training steps [16,18,23,24]. Compared to existing works, HACL successfully learns to get 2500 points within 33 million training frames without the aid of demonstrations. Moreover since this environment is full of traps, the success on this experiment confirm that despite the state graph contains misleading transitions, HACL still robustly learns to perform well.

4.1. Environments

4.1.1. Long-horizon chain

Following [21], we use a stochastic chain, but with longer chain length, to demonstrate the effectiveness of HACL in solving sparse reward problems. As shown in Fig. 4, the agent starts at s_1 and terminates at s_0. Action "Left" moves the agent left deterministically, but action "Right" moves the agent left or right randomly at 50%.

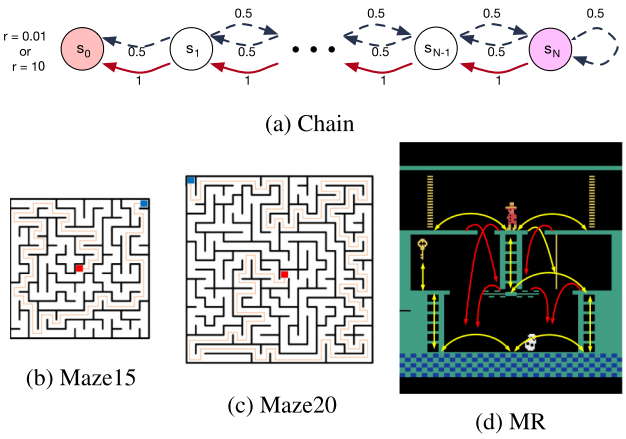


Fig. 4. Three Environments. (a) Chain: A long-horizon stochastic chain that requires massive exploration. (Dashed arrow: action "Right" moves the agent left or right randomly at 50%. Red arrow: action "Left" moves the agent left definitely.) (b) Maze15: size 15 x 15. (c) Maze20: size 20x20 (d) MR: The first room of Montezuma's Revenge. Some feasible routes are illustrated with yellow arrows, and the red ones are dead movements.

If the agent has visited s_N, the rightmost state, the reward of the episode is 10 otherwise 0.01. Note that the trivial reward 0.01 is easy to get, but traversing the long chain and returning demands a much longer action sequence, considering the random nature of the "Right" action. Different from the settings in [21], we scale the chain length from 6 to 17, adding much more difficulty.

Obviously, the optimal policy in this task is to head right till the rightmost state s_n and then step back to the terminal state s_0.

4.1.2. Grid maze

The grid maze environment is a large maze that contains far-away non-zero rewards. Fig. 4b and 4c shows two mazes of different sizes. The agent starts from the grid in red and terminates at the grid in blue. At each time step the agent moves to one of its neighboring cells by choosing among the discrete action space {UP, DOWN, LEFT, RIGHT}. If the agent knocks into the wall, no movement will be made. The agent gets zero rewards unless it reaches the terminal state. Given the large size of the maze, the reward is extremely sparse.

For this environment, the optimal policy is to walk along the path from the initial state to the terminal state.

4.1.3. Montezuma's revenge

Montezuma's Revenge (later abbreviated as MR) is a notoriously difficult game. In this game, the agent must navigate through different rooms to collect treasures. This environment is full of traps and pitfalls: falling from the platform will make the agent lose a life (Fig. 4d shows some dead movements in red and feasible movements in yellow).

4.2. Experimental settings

We compare HACL with the following baseline algorithms: two hierarchical reinforcement learning framework, HRL [21] and option-critic (later denoted as OC) [20], and another two reinforcement learning algorithms, DQN [1] and A3C [36]. HRL manually defined several landmarks in the first room of MR as the subgoals. But HACL makes spatial abstractions, automatically explores more macro-states and meanwhile generates the curricula. Likewise, option-critic does not require pre-defined hierarchy. It end-to-end learns the policies and terminations of options, as well as the policy over options. Also, we did ablative experiments to investigate the advantages of several innovations in HACL.

Please cite this article as: N. Jiang, S. Jin and C. Zhang, Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward, Neurocomputing, https://doi.org/10.1016/j.neucom.2019.06.024

Table 1
The number of frames each algorithm takes to perceive the optimal policy or get 2500 points in MR. The statistics are averaged over five random runs. The variants of HACL are introduced in Section 4.4. Note that, HACL*-NoCL requires a predefined hierarchical structure. For fair comparison, we select a good state graph, generated by one HACL trial to provide the hierarchy. This trial is denoted as HACL*. The chosen state graphs are with good quality (comparably fewer false transitions).

	A3C	DQN	HRL	Option-Critic	HACL-NoAbs	HACL-No2hl	HACL	HACL*	HACL*-NoCL
Chain			*inf*		**5M**	*inf*	7M	7M	*inf*
Maze15			40M+		13.1M	6.8M	**6.4M**	5.9M	7.1M
Maze20				60M+		16.3M	**15.2M**	12.2M	14.8M
MR				*inf*		46.4M	**33M**	23.4M	*inf*

Table 2
Performances of HACL and some state-of-the-art algorithms: HACL achieves comparable performance but with much fewer training frames. For HACL, its scores and frames are averaged over 5 random runs. Statistics of other algorithms are quoted from their original paper. Note that FuN's original paper [38] did not mention the average score. Its 2600 score is one best run. HIRL [18] gets 400 score with only 4M training frames. However, it utilizes imitation learning of expert demonstrations. Due to the space limit, H-No2hl and H-NoCL is short for HACL-No2hl and HACL-NoCL, respectively.

	Non-hierarchical						Hierarchical					
	DQN [1]	A3C [36]	R [15]	PC [23]	PCn [24]	Ape-X [16]	FuN* [38]	OC [20]	HIRL* [18]	HACL	H-No2hl	H-NoCL
Score	0	53	154	273.7	3705	2500	2600	0	400	2500	2500	0
Frames	200M	200M	200M	200M	150M	22.8B	-	200M	4M	33M	46.4M	60M

More experimental settings including the hyper-parameters, neural network architectures, state preprocessing in MR are introduced in Appendix B.

4.3. Performance

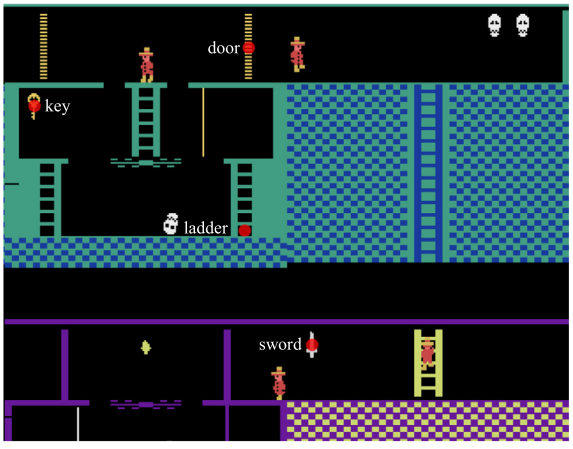
Compared to baseline algorithms, HACL and its variants achieve the best performance both in sample efficiency and final rewards on the three sparse reward tasks. Table 1 shows the number of training frames each algorithm take to perceive the optimal policy in the first two tasks. As to the challenging environment MR, it shows the training frames required to get 2500 points (This level is difficult).

For the long-horizon chain, when training from a random start, it is a remote possibility for the agent to keep heading to the right-most state. Thus, we observed that the baseline algorithms get stuck at the local optimal policy (directly move towards the left-most state to get trivial 0.01 reward) early and more training frames even strengthen this sub-optimal policy. So we marked training frames for these algorithms as *inf* in Table 1. However, HACL continuously provides dense pseudo rewards by the target-achieving reward, and accordingly, the agent itself is incentive enough to travel to the right-most state.

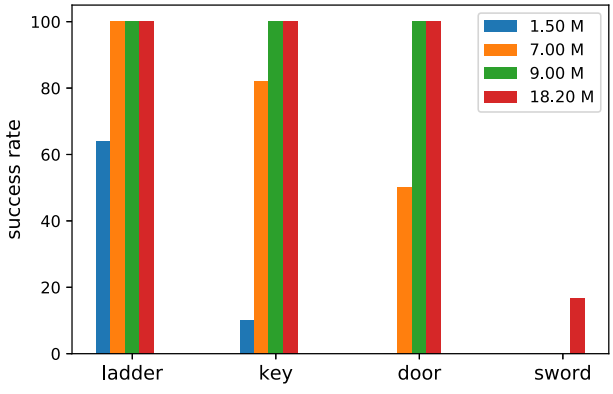
For the two maze tasks, we limit the total training frames to 40M and 60M, respectively. It can be easily seen from Fig. 4 that the agent needs to take nearly one hundred steps to the faraway reward state. Still, without informative exploration, all of the baseline algorithms fail to learn within the step limit.

On the sparse reward game, Montezuma's Revenge, Fig. 5 shows four key points of getting extrinsic rewards and the success rate of reaching these points at different training frames. The four key points are: climbing down two ladders to the ground, collecting the key(100 points), using the key to open the door(300 points), travelling faraway to the room bellow to get the sword(100 points). Within only 9M training steps, the agent learns to reach the first three points to get 400 points. However, as is shown in Fig. 5a, the long-horizon poses a great challenge to the agent to travel to the room below. It demands approximately 20M training steps to reach the sword. So a few existing works [18,21] get stuck at the 400 points. But in HACL, the dense target-achieving rewards and goal-reaching rewards progressively encourage the agent to travel to the room below to get the sword and more points.

Table 2 compares performances of HACL and some state-of-the-art algorithms on the Atari game Montezuma's Revenge. HACL



(a)



(b)

Fig. 5. (a) Four key points in game Montezuma's Revenge (b) the success rates of reaching four points at different training steps.

achieves 2500 points within 33 million training steps. Ape-X, though obtaining the same score, requires orders of magnitude more training steps than HACL. Even though the PCn learns to get higher rewards up to roughly 3700 points, they demand a huge number of training frames. Note that the score of HACL is averaged over 5 random runs. Its performance is stable across separate tri-

Please cite this article as: N. Jiang, S. Jin and C. Zhang, Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward, Neurocomputing, https://doi.org/10.1016/j.neucom.2019.06.024

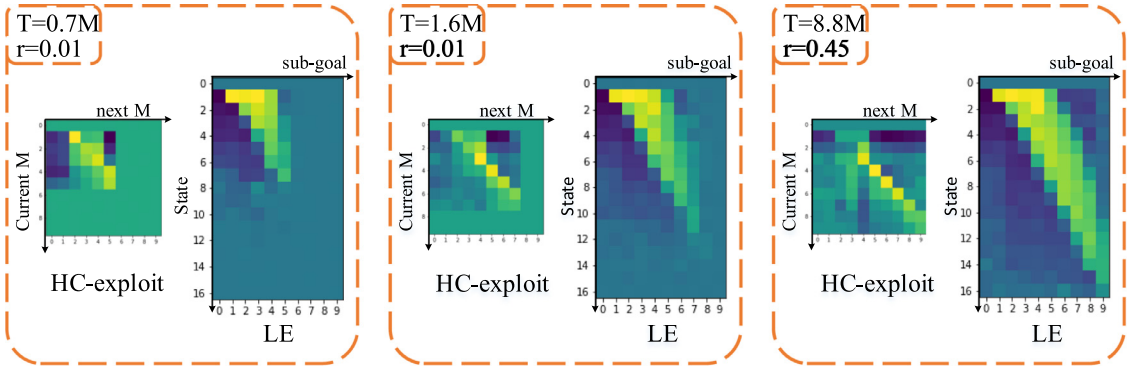


Fig. 6. visualization of the learning process of HACL-NoAbs, trained on a 17-state chain. Each orange block represents a timestamp: The training frames and average test reward are shown in the upper-left corner of each block. The square shows actor values of HC-exploit: given the current macro-state M_T, to get more extrinsic rewards, the action probability of the next macro-state M_{T+1}. The rectangle visualizes the actor values of low-level executors: given different subgoals, at each state, the probability of moving right. Brighter color represents higher values.

als. Due to constraints of computing time and resources, we could not train HACL with more than 60M training steps.

We compare HACL with its variants in the following ablative analysis.

4.4. Ablative analysis

To investigate the several advantages of HACL: (1) spatial-temporal abstraction (2) curriculum setting (3) introduction of HC-explore, we study the following variants of HACL: HACL-NoAbs, HACL-NoCL, HACL-No2hl, and HACL with a variable number of HC-explore workers. HACL-NoAbs makes each state as a macro-state, like the settings in the long-horizon chain experiment in [21]. HACL-NoCL does not generate target sets in a curriculum fashion but makes use of predefined subgoals. Manually setting subgoals and building hierarchy on this in advance is impossible when the agent needs to explore unknown environments. However, to investigate whether curriculum learning could accelerate the learning process, we employ some good state graphs (fewer false transitions) generated by HACL to provide the hierarchy. Also, we vary the number of HC-explore asynchronous threads in HACL to investigate the benefits of introducing HC-explore. If no HC-explores is used for training, the algorithm is denoted HACL-No2hl.

4.4.1. Macro-state abstraction

HACL-NoAbs is equivalent to set $\delta = 0$ in HACL, which means any two different states will be assigned to two different macro-states. The good result of HACL-NoAbs on the long-horizon chain shows HACL-NoAbs performs well when there are not too many states. In Chain, only 17 states exist in this MDP, modeling each state as a macro-state is feasible. Also because of the random nature of right action, abstracting more than one states into one macro-state poses challenge when navigating to those macro-states on the right. Hence, the macro-state abstraction does not function well in this task. While in Maze15 and Maze20, as the number of states increases, HACL-NoAbs consumes twice the number of training steps in Maze15, but fails in Maze20. As for MR, it is impossible to try HACL-NoAbs with numerous states. Therefore, this indicates that macro-state abstraction is critical for efficient exploration, especially for large state space.

4.4.2. Curriculum setting

HACL-NoCL makes use of predefined macro-states as subgoals, but no curriculum learning is employed. Table 1 compares HACL*-NoCL with HACL*. For fair comparison, the state graph generated by one HACL trial (this trial is denoted as HACL*) is selected to provide HACL*-NoCL with hierarchy. In chain, it is direct that without the curriculum training, the high-level conductor is prune to get stuck at the shorted-sighted local optimal policy: move left to get the 0.01 reward. For two maze environments, HACL*-NoCL learns the optimal policy but with more training steps than HACL*.

To fully understand the effect of curriculum learning, Fig. 6 visualizes tabular values of models at three different timesteps in Chain. It clearly shows the evolution of HACL's learning process: the agent continuously explores further, and the values are updated accordingly. At last, the agent "understands" it needs to migrate to the rightmost state to get the most reward. The low-level executor gives action "Right" if the subgoal is on its right, and vice versa. Fig. 6 demonstrates that, though the agent is prone to move left, HC-explore encourages the agent to explore further states gradually and finally perceive the optimal policy.

For MR, Table 2 shows HACL is able to obtain 2500 points within 33M frames, while HACL*-NoCL, with a better state graph, fails to get even 100 points within 60M training frames. The previous work [21] achieves 400 points with some predefined subgoals, which is similar to HACL*-NoCL. The reason why the performance of HACL*-NoCL is worse than that of [21] is possibly the large high-level action space in HACL*-NoCL. As it employs the full state graph generated in HACL*, the number of the high-level actions is much more than that in [21], which only manually defines 7 macro-states in the first room of MR.

Some false transitions in the state graph of Maze and MR do affect the curriculum sequence of the target set. But Section 5.1 will analyze that the consecutive stages of $\mathcal{T}$ still forms a curriculum sequence. Also, Section 5.3 argues that the imperfectness of the state graph will be remedied by high-level conductors and HACL still works robustly.

4.4.3. Functionality of two high-level conductors

Further, to demonstrate that the introduction of HC-explore accelerates training, we investigate the influence of the number of HC-explore training threads. The total number of asynchronous workers in A3C are maintained to be 6 in Chain and Maze. When using n workers for HC-explore, $6 - n$ workers will be trained with HC-exploit.

Fig. 7 illustrates the relationship between the number of HC-explore training workers and the total training steps HACL takes to converge to the optimal policy. The algorithm without the HC-explore is denoted as HACL-No2hl in Table 1. It is reasonable that the best choice of the number of HC-explore workers is a medium number, since replacing an HC-exploit worker to HC-explore marginally gives the agent a denser target-achieving reward, reducing the training frames, but with more HC-explore training workers, this marginal benefit gradually decreases and the

Please cite this article as: N. Jiang, S. Jin and C. Zhang, Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward, Neurocomputing, https://doi.org/10.1016/j.neucom.2019.06.024

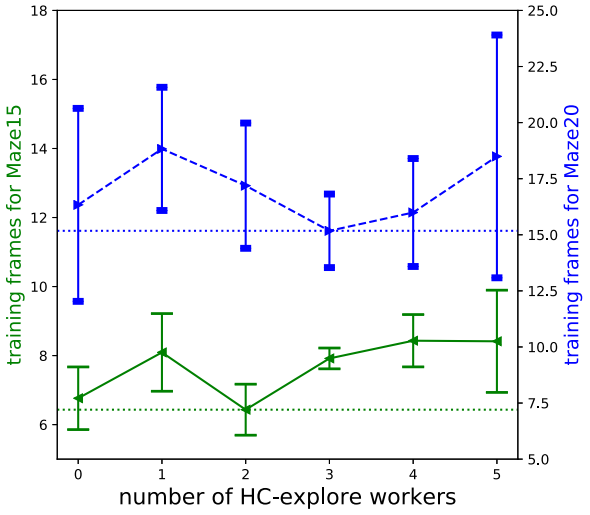


Fig. 7. The performance of HACL versus the number of HC-explore workers.

original objective of getting more extrinsic rewards may be obscured.

The result is the same to that in MR. HACL uses 10 workers, 9 for HC-exploit and 1 for HC-explore, whereas HACL-No2hl uses all the 10 workers with HC-exploit. As shown in Fig. 5a, the agent needs to take a much longer way to obtain the sword after getting 400 scores when opening the door. Therefore, without the dense target achieving reward provided by HC-explore, HACL-No2hl take longer to learn to get further scores due to the sparse reward nature of this game.

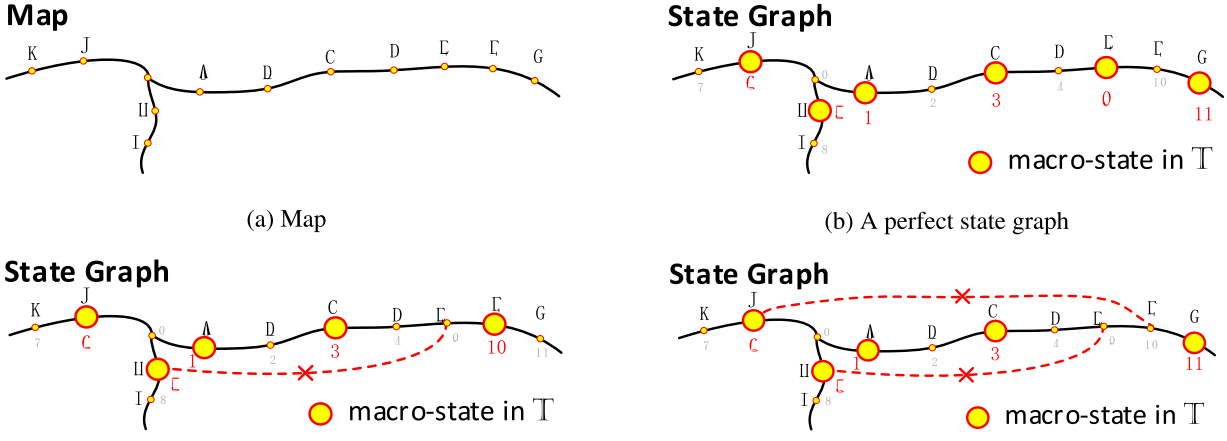
5. Analysis

During learning, the state graph stores the hierarchical structure of macro-state abstraction and generates a sequence of target sets. Therefore, it functions as a key role in HACL. In this section, we will first argue that: the generated sequence of target sets is always a curriculum sequence, even facing the false transitions in the state graph caused by the inaccuracy of the distance function. Then, we will discuss the principle in choosing a domain-specific hyper-parameter, the macro-state distance threshold δ, which is the root cause of false transitions. Finally, we give some arguments that in spite of the noisy state graph and flawed high-level plans in cold-start period, the curriculum setting and hierarchical reinforcement learning setting mitigates the negative effects of them. HACL still works robustly.

5.1. Curriculum sequence

The generated sequence of target sets is always a curriculum sequence, even with imperfect distance function. We will explain by Fig. 8, which plots several state graph sketches. More real state graphs in Maze and MR are listed in Fig. 9. Fig. 8a shows a map with true transitions between landmark positions. These landmarks are denoted with A, B, C,..., K, and each induces a macro-state when $\mathcal{G}$ is updated by the agent's trajectory. We denote their macro-state indexes as $I_A, I_B, \ldots, I_K$. The initial macro-state is marked with index 0. Algorithm 1 shows the order of the macro-state indexes accords with that of the agent's trajectory. Since the agent must visit *A, B, C, D, E, F* before *G*, it is guaranteed that $I_A < I_B < I_C < I_D < I_E < I_F < I_G$, likewise $I_H < I_I$, $I_J < I_K$. Therefore, when expanding the target set, all of the macro-states on the sure way from M_0 to it must have been considered to be added into $\mathcal{T}$, regardless of the false transitions. Fig. 8b shows an example of these macro-state indexes in the state graph. Their indexes are marked with gray numbers except the macro-states in $\mathcal{T}$ are marked with large dots and red numbers. Fig. 8b is a perfect state graph without false transitions, while Fig. 8c and 8d adds false transitions which is infeasible in practice but caused by the inaccuracy of the distance function. In all three state graphs, regardless of the existence of the false transitions, the sequence of the target set is always a curriculum sequence: When expanding the target set with a new macro-state i, the macro-states on the sure way from M_0 to M_i must have been considered to be added into $\mathcal{T}$.

However, how does the false transitions affect the curriculum sequence of $\mathcal{T}$? It becomes clear if comparing the difference between Fig. 8b, 8c, 8d. Algorithm 2 shows, a macro-state will be added into the target set only if it is not in the covered set of the existing target set. False transitions bring "blindly optimistic" covered set. Therefore, in Fig. 8c, $M_9(E)$ is expelled from adding into $\mathcal{T}$, and the difficulty gap between $M_3(C)$ and $M_{10}(F)$ enlarges. With more false transitions in Fig. 8d, both $M_9(E)$ and $M_{10}(F)$ are skipped and the gap further enlarges. In extreme cases, with many false transitions, the difficulty gap between consecutive stages of



(a) Map (b) A perfect state graph

(c) Imperfect state graph with one false transition (d) Imperfect state graph with more false transitions

Fig. 8. A map and three state graph sketches. The corresponding target set is marked with large dots and red numbers. Fig. 8b is a perfect state graph of the actual map in Fig. 8a. Some false transitions exist in Fig. 8c and 8d. The macro-states are added into the target set according to their indexes. E.g., the generated curriculum sequence of the target set in Fig. 8d is {1}, {1, 3}, {1, 3, 5}, {1, 3, 5, 6}, {1, 3, 5, 6, 11} if the adding count *K* in Algorithm 2 is 1.

Please cite this article as: N. Jiang, S. Jin and C. Zhang, Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward, Neurocomputing, https://doi.org/10.1016/j.neucom.2019.06.024

Table 3
The number of frames each HACL-δ takes to learn to get 2500 points in MR. $\|\mathcal{T}\|$ is the number of macro-states in $\mathcal{T}$ in the end of the training. The statistics are averaged by five random runs.

δ	16	18	20	22	24
training frames(M)	40.82	46.58	33.00	44.35	60M+
$\|\mathcal{T}\|$	28.6	25.8	22.2	20	19

the target set will be so huge that the benefit of curriculum learning diminishes.

5.2. Guides in choosing the threshold δ

HACL deals with the navigation-related tasks, so we simply choose the euclidean distance of the agent positions as the distance function. It could be applied in all the navigation-related tasks, as long as the agent's position is available. But there does exist a hyper-parameter which requires expertise knowledge and is crucial for learning, the distance threshold δ for adding a new macro-state.

Next, we talk about the principle in choosing this threshold and qualify the effect introduced by different choices of δ.

The hyper-parameter δ influences the construction of the state graph. Fig. 9 draws multiple state graphs and their corresponding target sets under different δ. Some false transitions are marked with red lines and crosses.

It is clear from Fig. 9 that with small δ, there will be more macro-states, but negligible false transitions. In this situation, benefiting from a high-quality state-graph and mild growth between consecutive stages of the target set, it is easier to train low-level policy at each curriculum stage. However, it brings some difficulty on the high-level plan, for the large amount of macro-states, and the learning process is quite slow. On the other hand, with large δ, the agent obtains more growth on the scope of navigation when progressing through each curriculum stage of $\mathcal{T}$. Both $\mathcal{G}$ and $\mathcal{T}$ will be concise, reducing the difficulty on the the high-level plan. However, in this situation, more false transitions emerges and lead to more aggressive covered set when expanding $\mathcal{T}$. Therefore, as analyzed in the previous section, large δ makes the difficulty level between consecutive stages of $\mathcal{T}$ grows too fast. The low-level executor may fail to learn due the long horizon, in which case the benefits of the curriculum learning gradually diminishes.

We suggest the following steps in choosing δ. First generate several state graphs and their corresponding target sets. Fig. 9a and 9b shows those of Maze15 and MR respectively. These graphs are obtained without much effort, just by 200 episodes of the agent's random walk. Then based on the analysis before, balancing the number of false transitions and high-level macro-states, a feasible range of δ could be determined. Further, experiments under different δ could be conducted to find the optimal choice of δ, or simply choose a moderate one.

From Fig. 9b, a feasible range of δ could be 16, 18, 20, 22, 24. We conducted experiments with all of them. HACL works robustly with $\delta = 16, 18, 20, 22,$ being able to learn to obtain 2500 points within 60M training frames, except for $\delta = 24$. The training frames required for obtaining 2500 points and the number of macro-states in $\mathcal{T}$ in the end of the training are tabulated in Table 3. All the statistics are averaged under five random runs. The result confirms our analysis before: larger δ makes the agent fail to navigate to positions farway and smaller δ brings more macro-states therefore posing difficulty on high-level plan. A moderate δ gives the best performance.

5.3. The robustness of HACL

Since there exist false transitions between macro-states, it is reasonable to question how the low-level agent behaves under the flawed high-level policy. Will the imperfect state graph and flawed high-level plan direct the whole learning process to collapse?

As to the imperfectness of the state graph, apart from its negative effect on the curriculum sequence, it has nothing to do with the high-level plan. As emphasized in Section 3.4, the high-level conductors learns by its intrinsic target-achieving reward to do high-level plans. This target-achieving reward teaches the high-level conductor, instead of the state graph.

Drawing on recursive proof in mathematics, we give a qualitative analysis that the hierarchical reinforcement learning setting and curriculum setting help mitigate the negative effects of flawed high-level policy, therefore make HACL more robust.

First, we must assume that the low-level learning algorithm is able to learn to reach places not too far, say within several δ. This is not a hard requirement, and it is reasonable to make such premise as we are marginally analyzing the effect of the flawed high-level plan. It could be achieved by choosing better low-level algorithm or smaller threshold δ. Now, we argue that, with the aid of hierarchical reinforcement learning setting and curriculum setting:

1. When in the stage $\mathcal{T}_0$, even working with a random high-level policy, the low-level agent could still manage to fulfill the condition required to progress to $\mathcal{T}_1$: be able to reach the macro-states in $\mathcal{T}_0$ with a predefined probability. Then the high-level policy is easy to learn. This process is shown in Fig. 10a. This argument is apparently established given the assumption on the low-level capability.
2. If the agent successfully progresses from $\mathcal{T}_{n-1}$ to $\mathcal{T}_n$, it is natural to assume the whole agent policy could give decent high-level plans and low-level decisions to reach macro-states in the covered set of $\mathcal{T}_{n-1}$. Then even the high-level agent is "partially random" with respective to the newly added macro-states, the low-level agent could still reach the newly added macro-states from the nearby covered places. The knowledge of navigating from the initial state to the covered state is reused, like the John's example in Introduction. In other words, this time, the newly added macro-states are no longer far away from the agent's initial position. The improved low-level policy in turn helps the training of high-level plan. Then, HACL manages to fulfill the condition required to progress to $\mathcal{T}_{n+1}$: be able to reach the macro-states in $\mathcal{T}_n$ with a predefined probability. This process is shown in Fig. 10b.

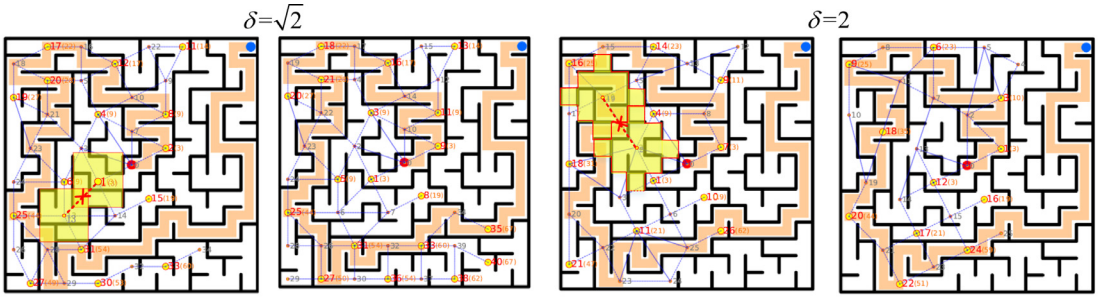
To evidence the arguments above, we modified HACL with a random high-level conductor, denoted as HACL-random. We must emphasize that, HACL-random differs from the settings in the above two arguments, in which the high-level plan is only flawed in the cold-start period but improves later with the help of an improved low-level policy. The result on the Maze15 and 20 confirms the arguments: the agent gradually learns to reach all of the macro-states in the state graph. As to the hard Montezuma's Revenge, we made four trials of HACL-random. Surprisingly, for twice, the agent is able to obtain the faraway sword (see Fig. 5a) within 60M training frames, which is the key to obtain 2500 points.

6. Related works

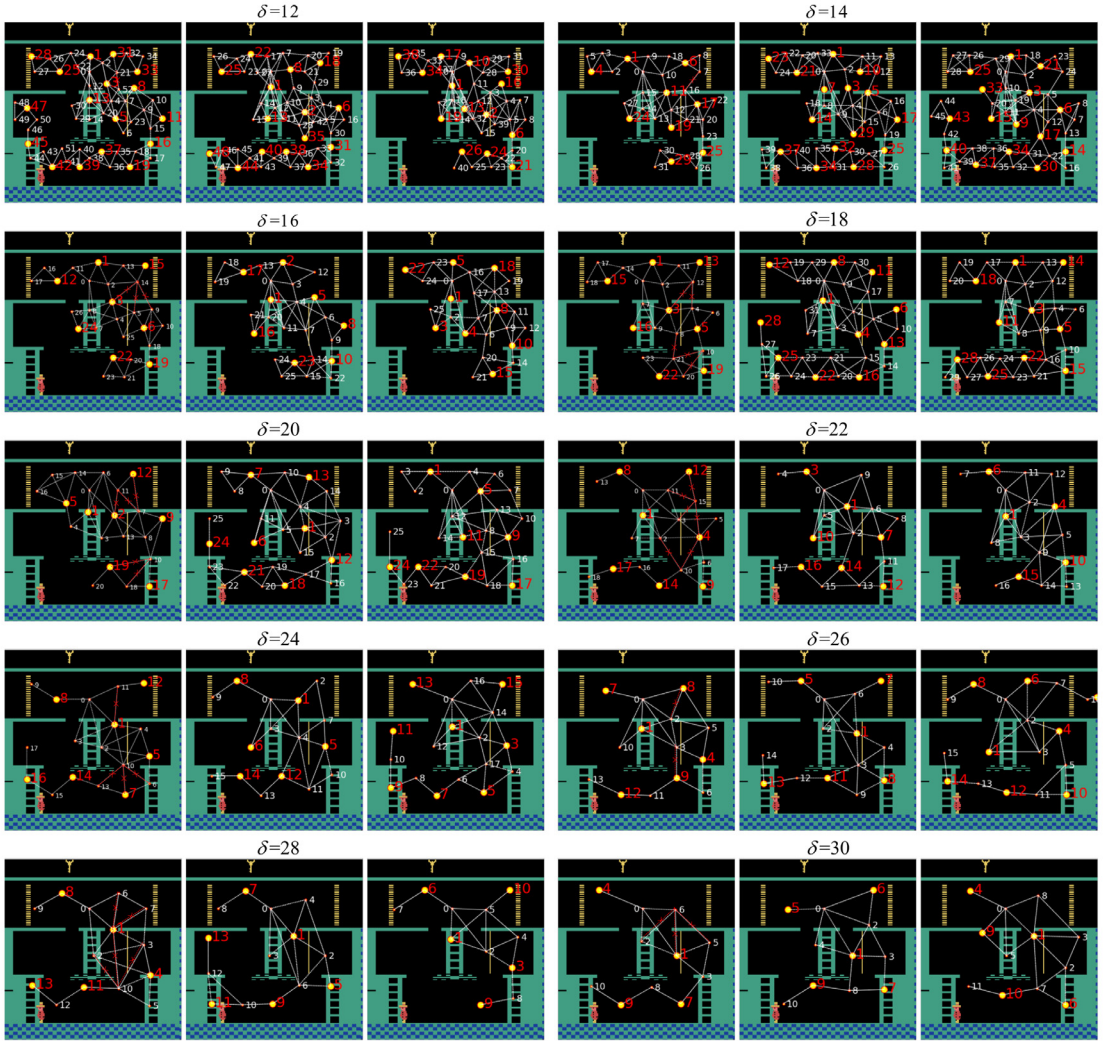
The proposed HACL lies at the intersection between hierarchical reinforcement learning and curriculum learning. Also, the introduction of pseudo target-achieving reward and goal-reaching reward relates to the works on intrinsic motivation.

Hierarchical reinforcement learning decomposes a complex problem into some interrelated sub-problems. By exploiting temporal and spatial abstractions, HRL is often used to address sparse reward tasks [11,18,21,25,27,28]. Some of these works require the

Please cite this article as: N. Jiang, S. Jin and C. Zhang, Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward, Neurocomputing, https://doi.org/10.1016/j.neucom.2019.06.024



(a) Maze15: The δ is $\sqrt{2}$ and 2 respectively. The distance of the macro-states in the target set to the initial state is shown after their indexes. It is better to zoom in to view. Two false transitions are highlighted with red lines and crosses. The Coverage of the macro-states at two ends of the false transitions are illustrated with yellow shadow.



(b) Montezuma's Revenge: The δ ranges from 12 to 30. The false transitions in the first state graph of each different δ are highlighted with red lines and crosses.

Fig. 9. Randomly generated state graphs and their corresponding target sets. Dotted lines are the transitions. Macro-states are indexed with gray or red numbers. The target set is marked with large dots and red numbers.

Please cite this article as: N. Jiang, S. Jin and C. Zhang, Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward, Neurocomputing, https://doi.org/10.1016/j.neucom.2019.06.024

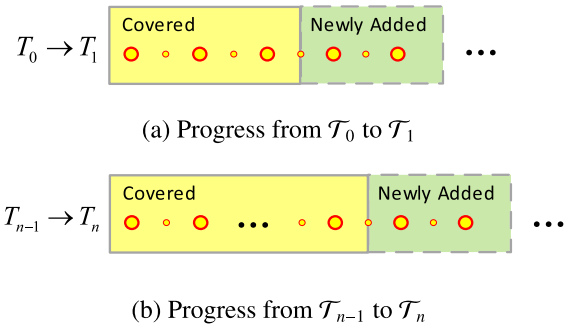


Fig. 10. Progress of HACL. Large dots represent the target set.

decomposable structure of the environment or manually designed subgoals, such that downstream tasks could be employed to learn skills. With the aid of the pre-defined hierarchy, some basic skills are trained in advance and reused in later complex tasks [21,27,28]. Some works do not require the decomposable structure or seek to discover the hierarchy automatically [11,18,20,38–42]. Gupta et al. [11] is the most similar to our work. It learns the confidence and belief about the environment, like the state graph in HACL, and gives hierarchical planning based on it. The key difference is that [11] learns confidence and belief about the environment while we build the state graph directly from the agent's trajectories. Moreover, its hierarchical planning is based on the value iteration network, while we learn the high-level and low-level policies by trial and error.

Algorithms with intrinsic motivation aim to encourage the agent to explore novel places [22–26,43]. These works design particular measurements to reflect the agent's curiosity of each state. This curiosity measurement then provides qualitative guidance for exploration. Also, there exist a few works making use of both HRL and intrinsic motivations to address the sparse reward problem [21,25].

Employing curriculum learning has been studied in some reinforcement learning tasks [10,30,32,44,45]. However, most of these works make use of handcrafted curricula. Further, [32–34,43] studied automatic curriculum generation in reinforcement learning. [34] addresses the sparse reward setting as well. It uses a Goal GAN to generate a curriculum sequence of goals which is similar to HACL. One difference is that HACL makes hierarchical actions. Also, [34] fails on the notoriously sparse reward game Montezuma's Revenge.[1]

7. Conclusion and future work

We have presented HACL, a hierarchical based automatic curriculum learning framework, and demonstrated its ability to alleviate the sparse reward difficulty. Our HACL framework consists of a two-level hierarchical decision model and a state graph for automatic curriculum learning. By employing curriculum learning, the sparse reward navigation challenge is alleviated by the dense target-achieving reward and goal-reaching reward. Moreover, unlike some existing works predefining subgoals in advance, the construction of curricula (the sequence of target sets) is fully automated, which automatically expands with the agent's capability. Combining the advantages of hierarchical reinforcement learning and curriculum learning, our method enables the agent to master three sparse reward tasks with much less training frame.

One defect of the current work is that we only employ a simple spatial distance between different places. This choice brings false transitions in the state graph and affects the quality of the curriculum sequence. The hierarchical setting and curriculum setting of HACL help mitigate its negative effects. But more domain knowledge could be introduced to design a more accurate distance function. As well, efficient incorporation of domain knowledge is beneficial for unknown challenging problems.

In this paper, we only investigate the navigation problem. Hence the state graph relates to the spatial information. However, the idea of automatic curriculum learning could be further extended to more general settings, e.g., macro-states in the state graph could present the difficulty level of the task, and the expansion of the target set indicates the increasing proficiency level of the agent on the task. Multiple directions could be studied in the future, like generalizing HACL to more general problems, learning the distance metric of macro-states, better incorporation of domain knowledge in the state graph construction, and the dynamic size of threshold δ.

Conflict of interest

None.

Acknowledgement

This work is supported by NSFC (Grant Nos. 61876095 and 61751308), and the Beijing Natural Science Foundation (Grant No. L172037).

Appendix A. Algorithm of HACL

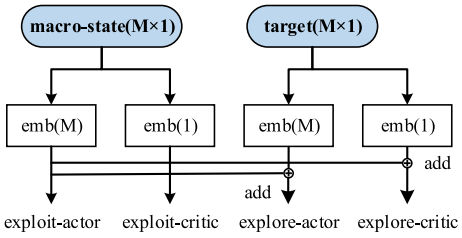
The training process of the whole HACL is tabulated in Algorithm 3. The hyper-parameters are introduced in Section 3.

Appendix B. Experimental settings

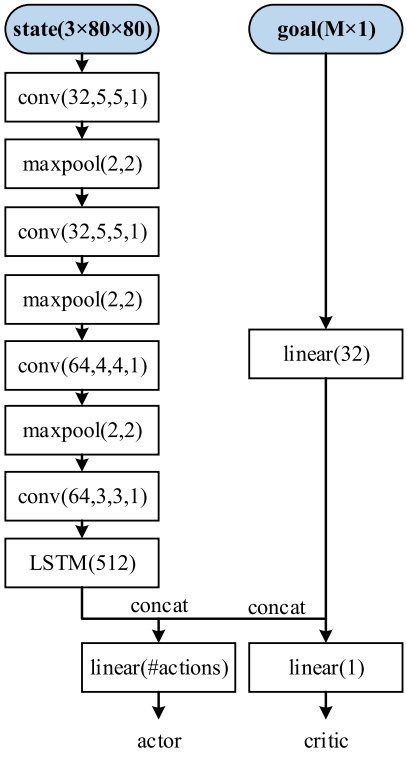
Hyper-parameters and training settings are listed in Table B.4. We keep most of the hyper-parameters the same, except those related to specific characteristics of the environment, like the low-level step limit *s*, the hyper-parameters about the state graph and the target set. We train HACL, with A3C using shared Adam optimizer with N workers, where N refers to the number of total workers. Other baseline algorithms are trained using their default settings, except that the number of workers used in A3C and the learning rates for all the baseline algorithms are the same to that in HACL. In HACL, some workers use HC-exploit as the high-level conductor, while the others use HC-explore. The impact of the number of HC-explore workers is introduced in Section 4.4. In all the three environments, we use the euclidean distance as the distance function. We obtain the global position of the agent's state to get such distance. The hyperparameter D refers to the maximum number of high-level decisions for each curriculum phase, which is introduced in Section 3.4.2. For Chain, we make it increase as an exponential of the current phase number *i*. As to Maze, which is an easy task, there is no need to set such a threshhold number.

Fig. B.11 shows the architecture of models for each tasks. *conv(c, w, h, s)* means a convolution layer with channel size *c*, kernel $w \times h$, and stride *s*. *maxpool(w, h)* represents a max-pooling layer with size $w \times h$. A nonlinear relu function is followed after all the convolution and linear layers, except the final output linear layer. *Emb(M)* is an embedding layer with output size M. The target input of the high-level conductor and subgoal input of the low-level executor are all one-hot vectors. In Fig. B.11, M refers to #total_macros in Table B.4 and Algorithm 3. The target set of macro-states is also the action space of the high-level conductor.

[1] We modified the released code https://github.com/florensacc/rllab-curriculum to test its performance on Montezuma's Revenge.



(a) high-level model for Chain



(b) low-level model for MR

Fig. B.11. Architectures of models in three tasks. M refers to the #total_macros in Table B.4.

However, the size of the target set n_t expands during learning. Here we set the corresponding layer size to be a maximum number M. During learning, even the input layer size and output layer size is a vector of length M, only the front n_t numbers of the subgoal input and target input will be non-zero, and the corresponding high-level action will only be selected from the first n_t elements of the high-level output. M is a predefined large number, and it has no effect on the learning process.

B.1. Long-horizon chain

In this task, the model of low-level executor contains tabular values of $A(s, g, a)$ for the actor branch, and $C(s, g)$ for the critic branch, in which s, g, a represent the state, subgoal, and action respectively. Fig. B.11a shows the model of high-level conductors. It is evident from the figure that the actor and critic values of HC-exploit are tabular as well.

Table B.4
Hyper-parameters of different tasks.

hyper-parameter	Chain	Maze15	Maze20	MR
threshold δ	0	$\sqrt{2}$		20
clip length l	0	0		30
m_1	2	6		6
K	1	3		2
n	2	6		6
p	0.2	0.5		0.4
D	$\{10000^*2^i\}$	*NAN*		2500
#total_workers, N	6	6		10
$\#\text{workers}_{HC-explore}$	4	2	3	1
optimizer	Adam			
α	0.5			
γ	0.99			
β	0.02			
c_1	2			
c_2	5			
c_3	0.1			0
hl lr	5E-5			
ll lr	5E-05			5E-06
ll step limit, s	40			300

B.2. Grid maze

In the grid maze, the agent has no access to the whole map. The same to the settings in long-horizon chain, the model of the low-level executor is tabular. For the ease of designing high-level models in different tasks, we simplify the high-level model as the tabular form as well. Experiments show that it achieves surprisingly good performance. The HC-explore maintains $A(t, m_1, m_2)$ for the actor branch, and $C(t, m_2)$ for the critic branch. HC-exploit maintains $A(m_1, m_2)$ for the actor branch, and $C(m_2)$ for the critic branch. t, m_1, m_2 represent the target, the last high-level action, the current high-level action, respectively.

B.3. Montezuma's Revenge

Fig. B.11b plots the model of low-level executor. The model of high-level conductors is the same to that in Grid Maze. The raw state is a RGB image with size 210x180. To emphasize the low-level subgoal, the position of the subgoal is highlighted with a white square. The top part of the image is cropped and the lower part with size 180×160 is scaled to 80×80 and normalized by a running mean and standard deviation.

Appendix C. An example state graph in Montezuma's Revenge

An example state graph and the target set in Montezuma's Revenge is shown in Fig. C.12. Interestingly, it is generated by the HACL-random, introduced in Section 5.3.

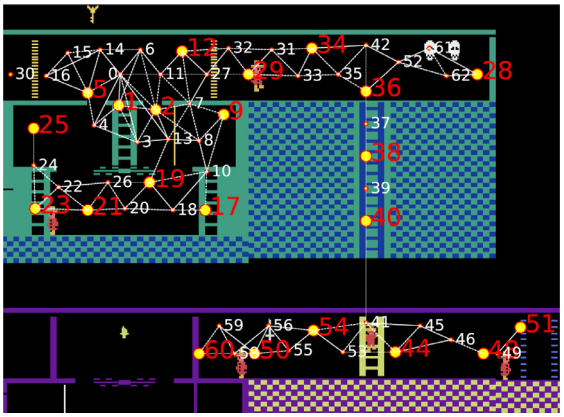


Fig. C.12. An example state graph and the target set in Montezuma's Revenge.

Please cite this article as: N. Jiang, S. Jin and C. Zhang, Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward, Neurocomputing, https://doi.org/10.1016/j.neucom.2019.06.024

References

[1] 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 518 (7540) (2015) 529–533.
[2] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489.
[3] S. Mathe, A. Pirinen, C. Sminchisescu, Reinforcement learning for visual object detection, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2894–2902.
[4] M. Bellver, X. Giro-i Nieto, F. Marques, J. Torres, Hierarchical object detection with deep reinforcement learning, in: Proceedings of the Deep Reinforcement Learning Workshop, NIPS, 2016.
[5] S.J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, V. Goel, Self-critical sequence training for image captioning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017.
[6] S. Liu, Z. Zhu, N. Ye, S. Guadarrama, K. Murphy, Improved image captioning via policy gradient optimization of spider, in: Proceedings of the IEEE International Conference Computer Vision, 3, 2017, p. 3.
[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv e-prints (2017). arXiv:1707.06347.
[8] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: Proceedings of the International Conference on Machine Learning, 2015, pp. 1889–1897.
[9] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A.J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, R. Hadsell, Learning to navigate in complex environments, in: Proceedings of the International Conference on Learning Representations, 2017.
[10] P. Mirowski, M.~K. Grimes, M. Malinowski, K.~M. Hermann, K. Anderson, D. Teplyashin, K. Simonyan, K. Kavukcuoglu, A. Zisserman, R. Hadsell, Learning to Navigate in Cities Without a Map, in: Proceedings of NIPS, 2018.
[11] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, J. Malik, Cognitive mapping and planning for visual navigation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017.
[12] Y. Zhu, R. Mottaghi, E. Kolve, J.J. Lim, A. Gupta, L. Fei-Fei, A. Farhadi, Target–driven visual navigation in indoor scenes using deep reinforcement learning, in: Proceedings of the ICRA, IEEE, 2017, pp. 3357–3364.
[13] D.L. Cruz, W. Yu, Path planning of multi-agent systems in unknown environment with neural kernel smoothing and reinforcement learning, Neurocomputing 233 (2017) 34–42.
[14] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning., in: Proceedings of the AAAI, 2016.
[15] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M.G. Azar, D. Silver, Rainbow: combining improvements in deep reinforcement learning, in: Proceedings of the AAAI, 2018.
[16] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, D. Silver, Distributed prioritized experience replay, in: Proceedings of the International Conference on Learning Representations, 2018.
[17] M.C. Machado, C. Rosenbaum, X. Guo, M. Liu, G. Tesauro, M. Campbell, Eigenoption discovery through the deep successor representation, in: Proceedings of the International Conference on Learning Representations, 2018.
[18] H.M. Le, N. Jiang, A. Agarwal, M. Dudk, Y. Yue, H. Daum III, Hierarchical imitation and reinforcement learning, in: Proceedings of the 35nd International Conference on Machine Learning, 2018.
[19] Y. Aytar, T. Pfaff, D. Budden, T. Le Paine, Z. Wang, N. de Freitas, Playing hard exploration games by watching YouTube, in: Proceedings of NIPS, 2018.
[20] P.-L. Bacon, J. Harb, D. Precup, The option-critic architecture., in: Proceedings of the AAAI, 2017.
[21] T.D. Kulkarni, K. Narasimhan, A. Saeedi, J. Tenenbaum, Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation, in: Proceedings of the NIPS, 2016.
[22] S.P. Singh, A.G. Barto, N. Chentanez, Intrinsically motivated reinforcement learning., in: Proceedings of the NIPS, 2004.
[23] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, R. Munos, Unifying count-based exploration and intrinsic motivation, in: Proceedings of NIPS, 2016.
[24] G. Ostrovski, M.G. Bellemare, A. van den Oord, R. Munos, Count-based exploration with neural density models, in: Proceedings of the 34nd International Conference on Machine Learning, 2017.
[25] C. Florensa, Y. Duan, P. Abbeel, Stochastic neural networks for hierarchical reinforcement learning, in: Proceedings of the International Conference on Learning Representations, 2017.
[26] D. Pathak, P. Agrawal, A.A. Efros, T. Darrell, Curiosity-driven exploration by self--supervised prediction, in: Proceedings of the 34nd International Conference on Machine Learning, 2017.
[27] C. Tessler, S. Givony, T. Zahavy, D.J. Mankowitz, S. Mannor, A deep hierarchical approach to lifelong learning in minecraft, in: Proceedings of the AAAI, 2017.
[28] N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, D. Silver, Learning and transfer of modulated locomotor controllers, arXiv e-prints (2016). arXiv:1610.05182.
[29] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26nd International Conference on Machine Learning, 2009.
[30] W. Zaremba, I. Sutskever, Learning to execute, in: Proceedings of the International Conference on Learning Representations, 2015.
[31] A. Graves, M.G. Bellemare, J. Menick, R. Munos, K. Kavukcuoglu, Automated curriculum learning for neural networks, in: Proceedings of the 34nd International Conference on Machine Learning, 2017.
[32] S. Narvekar, J. Sinapov, M. Leonetti, P. Stone, Source task creation for curriculum learning, in: Proceedings of the AAMAS, 2016.
[33] M. Svetlik, M. Leonetti, J. Sinapov, R. Shah, N. Walker, P. Stone, Automatic curriculum graph generation for reinforcement learning agents, in: Proceedings of the AAAI, 2017.
[34] C. Florensa, D. Held, X. Geng, P. Abbeel, Automatic goal generation for reinforcement learning agents, in: Proceedings of the 35nd International Conference on Machine Learning, 2018.
[35] R.S. Sutton, D. Precup, S. Singh, Between MDPS and semi-MDPS: a framework for temporal abstraction in reinforcement learning, Artif. Intell. 112 (1–2) (1999) 181–211.
[36] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: Proceedings of the International Conference on Machine Learning, 2016.
[37] L. Zuo, Q. Guo, X. Xu, H. Fu, A hierarchical path planning approach based on a and least-squares policy iteration for mobile robots, Neurocomputing 170 (2015) 257–266.
[38] A.S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, K. Kavukcuoglu, Feudal networks for hierarchical reinforcement learning, in: Proceedings of the 34nd International Conference on Machine Learning, 2017.
[39] M. Stolle, Automated discovery of options in reinforcement learning, McGill University, 2004 Ph.D. thesis.
[40] M.C. Machado, M.G. Bellemare, M.H. Bowling, A Laplacian framework for option discovery in reinforcement learning, in: Proceedings of the 34nd International Conference on Machine Learning, 2017.
[41] M.C. Machado, C. Rosenbaum, X. Guo, M. Liu, G. Tesauro, M. Campbell, Eigenoption discovery through the deep successor representation, in: Proceedings of the International Conference on Learning Representations, 2018.
[42] K. Frans, J. Ho, X. Chen, P. Abbeel, J. Schulman, Meta Learning Shared Hierarchies, in: Proceedings of the International Conference on Learning Representations, 2018.
[43] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, R. Fergus, Intrinsic motivation and automatic curricula via asymmetric self-play, in: Proceedings of the International Conference on Learning Representations, 2018.
[44] A. Vezhnevets, V. Mnih, S. Osindero, A. Graves, O. Vinyals, J. Agapiou, k. kavukcuoglu, Strategic attentive writer for learning macro-actions, in: Proceedings of NIPS, 2016.
[45] Z. Ren, D. Dong, H. Li, C. Chen, Self-paced prioritized curriculum learning with coverage penalty in deep reinforcement learning, IEEE Trans. Neural Netw. Learn. Syst. 29 (6) (2018) 2216–2226.

Nan Jiang received the B.S. degree from the Department of Physics, Tsinghua University, China, in 2015 and currently a Ph.D. candidate at the State Key Laboratory of Intelligent Technologies and Systems, Department of Automation, Tsinghua university, Beijing China. Her research interests include deep reinforcement learning, hierarchical reinforcement learning, computer vision.

Sheng Jin received the B.S. degree from the Department of Automation, Tsinghua University, China in 2017 and currently a Master student at the State Key Laboratory of Intelligent Technologies and Systems, Department of Automation, Tsinghua University. His interests include deep reinforcement learning, computer vision.

Changshui Zhang received the B.S. degree in mathematics from Peking University, Beijing, China, in 1986 and the M.S. and Ph.D. degrees in control science and engineering from Tsinghua University, Beijing, in 1989 and 1992, respectively. Since 1992, he has been with the Department of Automation, Tsinghua University, where he is currently a Professor. He is the author of more than 200 papers. His research interests include pattern recognition and machine learning.

Dr. Zhang is a member of the Standing Council of the Chinese Association of Artificial Intelligence. He currently serves as an Associate Editor of Pattern Recognition Journal.

Please cite this article as: N. Jiang, S. Jin and C. Zhang, Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward, Neurocomputing, https://doi.org/10.1016/j.neucom.2019.06.024