Construction of Macro Actions for Deep Reinforcement Learning

Abstract

Conventional deep reinforcement learning typically determines an appropriate primitive action at each timestep, which requires enormous amount of time and effort for learning an effective policy, especially in large and complex environments. To deal with the issue fundamentally, we incorporate macro actions, defined as sequences of primitive actions, into the primitive action space to form an augmented action space. The problem lies in how to find an appropriate macro action to augment the primitive action space. The agent using a proper augmented action space is able to jump to a farther state and thus speed up the exploration process as well as facilitate the learning procedure. In previous researches, macro actions are developed by mining the most frequently used action sequences or repeating previous actions. However, the most frequently used action sequences are extracted from a past policy, which may only reinforce the original behavior of that policy. On the other hand, repeating actions may limit the diversity of behaviors of the agent. Instead, we propose to construct macro actions by a genetic algorithm, which eliminates the dependency of the macro action derivation procedure from the past policies of the agent. Our approach appends a macro action to the primitive action space once at a time and evaluates whether the augmented action space leads to promising performance or not. We perform extensive experiments and show that the constructed macro actions are able to speed up the learning process for a variety of deep reinforcement learning methods. Our experimental results also demonstrate that the macro actions suggested by our approach are transferable among deep reinforcement learning methods and similar environments. We further provide a comprehensive set of ablation analysis to validate our methodology.

1 Introduction

Conventional deep reinforcement learning (DRL) has been shown to demonstrate superhuman performance on a variety of environments and tasks (Mnih et al., 2013, 2015, 2016; Salimans et al., 2017; Schulman et al., 2017; Moriarty et al., 1999; Such et al., 2018). However, in conventional methods, agents are restricted to make decisions at each timestep, which differs much from the temporally-extended framework of decision-making in human beings. As a consequence, traditional methods (Mnih et al., 2013, 2016; Houthooft et al., 2016) require enormous amounts of sampling data in environments where goals are hard to reach or rewards are sparse. In complex environments where goals can only be achieved by executing a long sequence of primitive actions, it is difficult to perform exploration efficiently. As most real-world environments are large, complex, and usually offer sparse rewards, finding an optimal policy is still hard and challenging. It becomes crucial to explore new mechanisms to deal with these environments more efficiently and effectively.

Researchers in the past few years have attempted various techniques to expand the realm of DRL to *temporally-extended frameworks* (Sutton et al., 1999; Vezhnevets et al., 2016; Kulkarni et al., 2016; Bacon et al., 2017; Frans et al., 2017; Daniel et al., 2016; Florensa et al., 2017; Machado et al., 2017). In such frameworks, a high-level controller interacts with the environment by selecting temporal-extended policies usually named as “options”. Once an option is selected, it interacts with the environment for a certain timesteps and perform primitive actions until a termination condition for that option is met. However, developing effective options either requires a significant amount of domain knowledge (Girgin et al., 2010), or often restricted to low-dimensional and/or relatively simple environments only (Bacon et al., 2017; Heess et al., 2016; Kulkarni et al., 2016).

Instead of developing options, another branch of research directions focus on constructing macro actions (Fikes and Nilsson, 1971; Siklossy and Dowson, 1977; Minton, 1985; Pickett and Barto, 2002; Botea et al., 2005; Newton et al., 2005, 2007). A macro action (or simply "*a macro*") is an open-loop (DiStefano III et al., 1967) policy composed of a finite sequence of primitive actions. Once a macro is chosen, the actions will be taken by the agent without any further decision making process. Some researches in DRL attempt to construct macros from the experience of an agent (Durugkar et al., 2016; Randlov, 1999; Yoshikawa and Kurihara, 2006; Onda and Ozawa, 2009; Garcia et al., 2019). A key benefit of these approaches is the ease to construct a desired macro without supervision (Durugkar et al., 2016). However, these approaches may lead to biased macros. For example, the most frequently used sequence of actions may not correspond to a macro that can lead the agent to outperform its past policies. Furthermore, as agents generally perform exploration extensively in the early stages of training, the inconsistency in the early experience may perturb the construction of macros. A few researchers proposed to employ a reduced form of macro called *action repeat* (Lakshminarayanan et al., 2017; Sharma et al., 2017). In this formulation, primitive actions are repeated several times in a macro before the agent makes another decision. However, this formulation may limit the diversity of macros. By relaxing the agent to perform macros consisting of diversified actions, the agent is granted more chances to achieve higher performance. In addition, there are a handful of researches that requires human supervision to derive macros for improving training efficiency. The authors in McGovern et al. (1997) show that handcrafted macros can speed up training in certain tasks but hinder performance in others. The authors in Heecheol et al. (2019) generate macros from expert demonstrations via a variational auto-encoder. However, the process of obtaining such demonstrations is expensive. It would thus be favorable if there exists a method to find a macro without human intervention. Nevertheless, little attention has been paid to the construction of such macros.

Our goal is to develop a methodology for constructing a macro action from possible candidates. As possible macros are allowed to have different lengths and arbitrary compositions of primitive actions, such diversified macro actions essentially form an enormous space. We define this space as the *macro action space* (or simply "*macro space*"). Repeated action sequences are simply a small subset of the macro space. For a specific task in an environment, we hypothesize that there are good macros and bad macros in the macro space. Different macro actions have different performance impacts to an agent. Good macro actions enable the agent to jump over multiple states and reach a target state quicker and easier. On the other hand, bad macro actions may lead the agent to undesirable states. We argue that whether a macro is good or bad can only be determined by direct evaluation. In this study, we propose an evaluation method to test whether a macro is satisfactory for an agent to perform a specific task in an environment. Our method first relaxes the conventional action space (Sutton and Barto, 2018) with a macro to form an *augmented action space*. We then equip the agent with the augmented action space, and utilize the performance results as the basis for our evaluation. In order to find a good macro in the vast macro space, a systematic method is critically important and necessary. The method entails two prerequisites: a macro construction mechanism and a macro evaluation method. Although the second one is addressed above, there is still a lack of an appropriate approach to construct macros.

To satisfy the above requirement, we embrace an *genetic algorithm* (or simply "*GA*") for macro construction. GA offers two promising properties. First, it eliminates the dependency of the macro action derivation procedure from the past policies of an agent and/or human supervision. Second, it produces diversified macros by mutation. In order to combine GA with our evaluation method, our approach comprises of three phases: (1) macro construction by GA; (2) action space augmentation; and (3) evaluation of the augmented action space. Our augmented action space contains not only the original action space defined by DRL, but also the macro(s) constructed by GA. To validate the proposed approach, we perform our experiments on *Atari 2600* (Brockman et al., 2016) and *ViZDoom* (Kempka et al., 2016), and compare them to two representative DRL baseline methods. We demonstrate that our proposed method is complementary to existing DRL methods, and perform favorably against the baselines. Moreover, we show that the choice of the macro have a crucial impact on the performance of an agent. Furthermore, our results reveal the existence of transferability of a few macros over similar environments or DRL methods. We additionally provide a comprehensive set of ablation analysis to justify various aspects of our approach. The contributions of this paper are summarized as follows:

- We define the proposed approach as a framework.
- We provide a definition of macro action space.
- We introduce an augmentation method for action spaces.
- We propose an evaluation method to determine whether a macro is good or not.
- We establish a macro action construction method using GA for DRL.
- We investigate and reveal the transferability of macro actions.

The rest of this paper is organized as follows. Section 2 explains our framework. Section 3 describes our implementation details. Section 4 presents our results. Section 5 concludes.

2 Framework Formulation

In this section, we first provide the definition of macro actions. Then, we provide a model of the environment with macros, which is a special case of Semi-Markov Decision Processes (SMDP) (Sutton et al., 1999). Next, we provide definitions of functions in DRL. Finally, we formulate a framework for constructing good macros. The essential notations used in this paper can be referenced in Table A1 in our appendices.

Macro action. A macro action is defined as a finite sequence of primitive actions $m = (a_1, a_2, \ldots, a_k)$, for all a_i in action space $\mathcal{A}$, and some natural number k. Macros can be selected atomically as one of the actions by an agent. The set of macros form a macro action space $\mathfrak{M}$, which can be represented as $\mathfrak{M} = \mathcal{A}^+$, where '+' stands for *Kleene plus* (Hopcroft and Ullman, 1979).

Environment modeled. The environment we concerned can be modeled as a special case of SMDP, which can be represented as a 4-tuple $(\mathcal{S}, \mathcal{M}, p^m_{ss'}, r^m_s)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{M}$ the finite set containing macro and all primitive actions provided by the environment, $p^m_{ss'}$ the transition probability from s to s' when executing m, and r^m_s the reward received by the agent after executing m. The expressions of $\mathcal{M}$, $p^m_{ss'}$, and r^m_s can be represented as Eqs. 1, 2, and 3, respectively.

Deep reinforcement learning. An agent interacts with an environment under a policy ν, where ν is a mapping, $\nu : \mathcal{S} \times \mathcal{M} \to [0, 1]$. The expected cumulative rewards it receives from each state s under ν can be denoted as $V^\nu(s)$. The objective of DRL is to train an agent to learn an optimal policy such that it is able to maximize its expected return. The maximal expected return from each state s under the optimal policy can be denoted as $V^*_{\mathcal{M}}(s)$. The expressions of V^ν and $V^*_{\mathcal{M}}$ can be represented as Eqs. 4 and 5, respectively, where $\gamma \in [0, 1]$. We only use γ between macros but not between the primitive actions within a macro. This encourages the agent to prefer the provided macros over a series of primitive actions.

$$\mathcal{M} = \mathcal{A} \cup \{m\}, \text{ where } m \in \mathfrak{M} \tag{1}$$

$$p^m_{ss'} = \mathbb{P}\left\{s_{t+|m|} = s' \middle| s_t = s, m_t = m\right\} \tag{2}$$

$$r^m_s = \mathbb{E}\left\{\sum_{\tau=0}^{|m|-1} r_{t+\tau} \middle| s_t = s, m_t = m\right\} \tag{3}$$

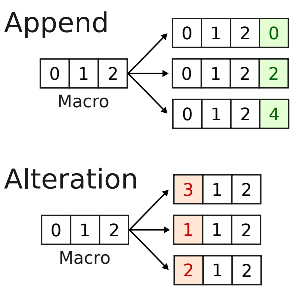
$$V^\nu(s) = \sum_{m \in \mathcal{M}} \nu(s, m) \left[r^m_s + \gamma \sum_{s' \in \mathcal{S}} p^m_{ss'} V^\nu(s')\right] \tag{4}$$

$$V^*_{\mathcal{M}} = \max_{m \in \mathcal{M}} \left[r^m_s + \gamma \sum_{s' \in \mathcal{S}} p^m_{ss'} V^*_{\mathcal{M}}(s')\right] \tag{5}$$

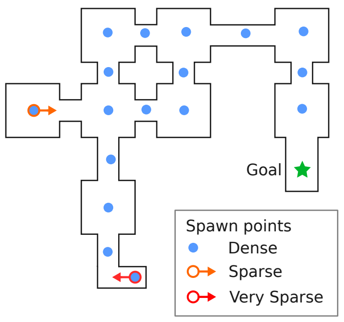
Framework. We define our framework as a 4-tuple $(\mathscr{R}, \mathscr{C}, \mathscr{A}, \mathscr{E})$, where $\mathscr{R}$ is the collection of all DRL methods, $\mathscr{C}$ the collection of all macro action construction methods, $\mathscr{A}$ the collection of all action space augmentation methods, and $\mathscr{E}$ the collection of all evaluation methods. Following this framework, in this study, we select Proximal Policy Optimization (PPO) (Schulman et al., 2017) and Advantage Actor Critic (A2C) (Mnih et al., 2016) as our DRL methods. Our implementations of the latter three components of the 4-tuple are formulated as pseudocodes and are presented in Algorithms 1, A1, and A2, respectively.

Algorithm 1 Macro construction algorithm based on GA

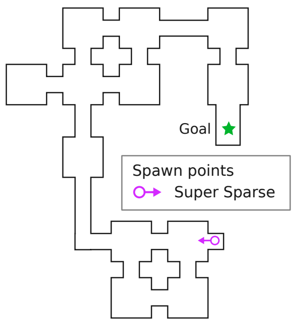
```
 1: input: Environment $\mathcal{E}$; DRL algorithm $\mathcal{R}$; Total number of evaluated macros $k$
 2: input: The sizes of sets $Q, Q_+, Q_* = q, q_+, q_*$ respectively
 3: output: List of the top $q$ highest-performing macros evaluated $Q$
 4: function CONSTRUCTION($\mathcal{E}$, $\mathcal{R}$, $k$, $q$, $q_+$, $q_*$)
 5:     initialize: $\mathcal{A}$ = the primitive action space of $\mathcal{E}$
 6:     initialize: $M = [m_1, \dots, m_q]$, a list of $q$ randomly generated macros s.t. $\forall\ m \in M, |m| = 2$
 7:     initialize: $F = [f_1, \dots, f_q], \forall\ f \in F, f = 0$, the fitness scores for all the macros in $M$
 8:     initialize: $Q = \emptyset, Q_+ = \emptyset, Q_* = \emptyset$, $i = 0$   ▷ $Q$ contains the top $q$ macros at any generation
 9:     while $i < k$ do                                   ▷ Each iteration is one generation
10:         for $j$ in 1 to $q$ do
11:             $\mathcal{M}$ = AUGMENT($\mathcal{A}, m_j$)            ▷ Please refer to Algorithm A1
12:             $f_j$ = EVALUATE($\mathcal{E}$, $\mathcal{R}$, $\mathcal{M}$)          ▷ Please refer to Algorithm A2
13:             $i = i + 1$
14:             if $i >= k$ then break
15:         end for
16:         $Q$ = the top $q$ scoring macros in $Q \cup [m_1, \dots, m_q]$ according to $F$
17:         for $m$ in List of $q_+$ randomly selected macros in $Q$ do
18:             $Q_+ = Q_+ \cup \{$APPEND$(\mathcal{A}, m)\}$          ▷ Please refer to Algorithm A3
19:         end for              ▷ $Q_+$ will hold $q+$ mutated macros using append operator
20:         for $m$ in List of $q_*$ randomly selected macros in $Q$ do
21:             $Q_* = Q_* \cup \{$ALTER$(\mathcal{A}, m)\}$           ▷ Please refer to Algorithm A4
22:         end for              ▷ $Q_*$ will hold $q*$ mutated macros using alternation operator
23:         $M = Q_+ \cup Q_*$
24:     end while
25:     return $Q$
26: end function
```



(a) The genetic operators.



(b) The map for the *Dense*, *Sparse*, and *Very Sparse* tasks.



(c) The map for the *Super Sparse* task.

Figure 1: (a) The generic operators for the append and alteration operations in Section 3. (b)(c) Map layouts of the *ViZDoom* environment. The blue points denote the random spawn points for the *Dense* task, while the orange, red, and purple arrows denote the fixed spawn points of the agents and their initial orientations for the *Sparse*, *Very Sparse*, and *Super Sparse* tasks, respectively.

3 Generic Algorithm for Constructing Macro Actions

In this section, we present our implementation details of GA for constructing macro actions. We formulate our macro construction algorithm based on GA as Algorithm 1, which is established atop (1) the action space augmentation method (presented as Algorithm A1 and accessed at line 11 of Algorithm 1), (2) the macro evaluation method (presented as Algorithm A2 and accessed at line 12 of Algorithm 1), as well as (3) the append operator (accessed at line 19) and (4) the alteration operator (accessed at line 22) presented in Fig. 1 (a) and the appendices for mutating the macros.

We walk through Algorithm 1 and highlight the four phases of GA as follows. Lines 1-3 declare the input and output of the algorithm. Lines 5-8 correspond to the "*initialization phase*", which initializes the essential parameters. The population of the macros M are initialized at line 6. Lines 10-15 correspond to the "*fitness phase*", which augments the action space and performs fitness evaluation. The *fitness phase* can be executed in parallel by multiple threads. Lines 16-17 corresponds to the "*selection phase*", which selects the top performers from the population. Lastly, lines 18-23 correspond to the "*mutation phase*", which alters the selected macros and updates the population.

4 Experimental Results

In this section, we present our experimental results of Algorithm 1 and discuss their implications. We arrange the organization of our presentation as follows in order to validate the methodology we proposed in the previous sections. First, we walk through the experimental setups. Second, we compare the constructed macro actions of different generations to examine the effectiveness of Algorithm 1. Third, we investigate the compatibility of the constructed macros with two off-the-shelf DRL methods. Then, we explore the transferability of the constructed macros between the two DRL methods. We next demonstrate that the constructed macros can be transferred to harder environments. Finally, we present a set of ablation analysis to inspect the proposed methodology from different perspectives. Please note that we only present the most representative results in this section. We strongly suggest the interested readers to refer to our appendices for more details.

4.1 Experimental Setup

We first present the environments used in our experiments, followed by a brief description of the baselines adopted for comparison purposes. All of the macros presented throughout this section are constructed by Algorithm 1, if not specifically specified. We tabularize the environmental configurations, the hyper-parameters of Algorithm 1, and the hyper-parameters used during the training phase in our appendices. Except for Section 4.2, all of the curves presented in this section are generated based on five random seeds and drawn with 95% confidence interval (displayed as the shaded areas).

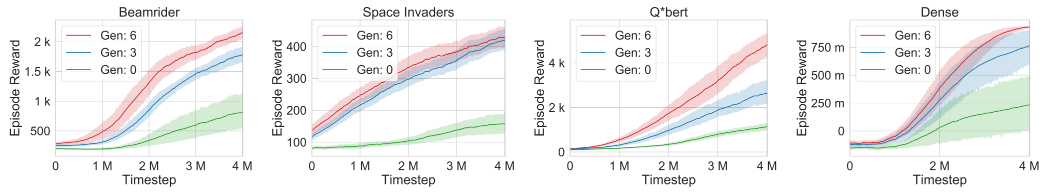
Environments. We employ two environments with discrete primitive action spaces in our experiments: *Atari 2600* (Bellemare et al., 2013) and *ViZDoom* (Wydmuch et al., 2018). For *Atari 2600*, we select seven representative games to evaluate our methodology: *BeamRider*, *Breakout*, *Pong*, *Q*bert*, *Seaquest*, *SpaceInvaders*, and *Enduro*. Due to the limited space, we use only the former six games for presenting our comparisons and analyses in Section 4, while leaving the remainder of our results in the appendices. For *ViZDoom*, we evaluate our methodology on the default task `my_way_home` (denoted as "*Dense*") for comparing the macros constructed among generations. We further use the "*Sparse*", "*Very Sparse*" and, "*Super Sparse*" (developed by us) tasks for analyzing the properties of compatibility and transferability of the constructed macros mentioned above. The *Super Sparse* task comes with extended rooms and corridors in which the distance between the spawn point of the agent and the goal is farther than the other three tasks. The map layouts for these four tasks are depicted in Figs. 1 (b) and 1 (c).

Baselines. In our experiments, we select PPO (Schulman et al., 2017) and A2C (Mnih et al., 2016) as our DRL baselines for training the agents. For the *ViZDoom* environment, we further incorporate the intrinsic curiosity module (ICM) (Pathak et al., 2017) in the *Sparse*, *Very Sparse*, and *Super Sparse* tasks to generate intrinsic rewards for motivating exploration.

***ViZDoom* setups for transferability evaluation.** We use *ViZDoom* to validate our hypothesis that macro actions constructed from an easy environment can be transferred to complex environments. In order to perform this validation study, we use Algorithm 1 to construct macro actions from the *Dense* task and evaluate their performance after 5M training timesteps. The highest performing macro action is then selected and evaluated in the *Sparse*, *Very Sparse*, and *Super Sparse* tasks respectively after 10M training timesteps.

4.2 Comparison of Macros among Generations

Fig. 2 compares the learning curves of the A2C agents trained with different generations of the constructed macros in *BeamRider*, *SpaceInvaders*, *Q*bert*, and *Dense*. Each curve along



(a) *BeamRider*. (b) *SpaceInvaders*. (c) *Q*bert*. (d) *Dense*.

Figure 2: Learning curves of the A2C agents trained with the early, middle, and late generations of the constructed macros. Please note that "Gen" stands for "generation" for all of the four figures.

Table 1: Evaluation results of *Atari*. We calculate the mean episode rewards over 100 episodes.

Environment	A2C	A2C+$\hat{m}_{\text{A2C}}$	$\hat{m}_{\text{A2C}}$	**Enhancement**
BeamRider	1049.52	2796.20	$(1, 0, 1, 1)$	**166.43%**
Breakout	81.76	119.36	$(3, 3, 0, 2)$	**45.99%**
Enduro	0	1270.37	$(1, 1)$	**NaN**
Pong	4.54	20.47	$(2, 0, 2)$	**350.88%**
*Q*bert*	2411.50	4540.25	$(0, 2, 4, 4)$	**88.28%**
Seaquest	420.00	657.80	$(2, 3, 4, 0)$	**56.62%**
SpaceInvaders	214.55	454.25	$(1, 0, 0)$	**111.72%**

with the corresponding confidence interval stands for the average performance of the best macros in the generation (i.e., Q in Algorithm 1). It is observed from the trends that the mean episode rewards obtained by the agents improve with generations, revealing that later generations of the constructed macros may outperform earlier ones. The improving trends suggest that the evaluation method of Algorithm A2 is effective and reliable for Algorithm 1.

4.3 Compatibility of the Constructed Macros with DRL Methods

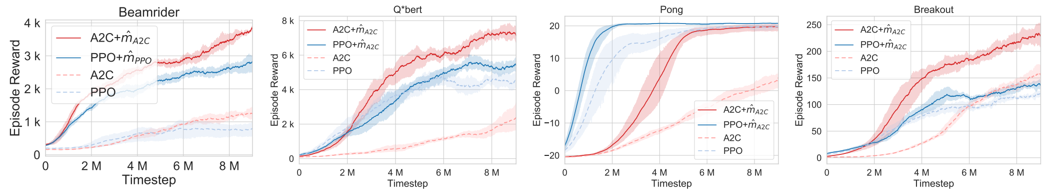
In order to examine whether the best macros constructed by our methodology is complementary to the baseline DRL methods, we first execute Algorithm 1 with A2C and PPO, and determine the best macros $\hat{m}_{\text{A2C}}$ and $\hat{m}_{\text{PPO}}$ for the two DRL baselines respectively. We then form the augmented action spaces $\mathcal{M}_{\text{A2C}} = A \cup \{\hat{m}_{\text{A2C}}\}$ and $\mathcal{M}_{\text{PPO}} = A \cup \{\hat{m}_{\text{PPO}}\}$. We next train A2C using $\mathcal{M}_{\text{A2C}}$ and PPO using $\mathcal{M}_{\text{PPO}}$ for 10M timesteps respectively. The results evaluated on *BeamRider* are shown in Fig. 3 (a). The learning curves reveal that the baseline DRL methods are both significantly benefited from the augmented action spaces. More supporting evidences are presented in Figs. 3 and 4. Table 1 presents our evaluation results for the seven Atari games. It is observed that all of the constructed macros are able to considerably enhance the performance of the baseline DRL methods, especially for *Enduro*.

4.4 Transferability of the Constructed Macros among DRL Methods

In order to inspect whether the macro constructed for one DRL baseline can be transferred to and utilized by another DRL baseline, we train PPO with $\mathcal{M}_{\text{A2C}}$ for 10M timesteps and plot the results on *Q*bert*, *Pong*, and *Breakout* in Figs. 3 (b), 3 (c), and 3 (d), respectively. The results show that PPO is able to use $\hat{m}_{\text{A2C}}$ to enhance both the performance and the learning speed of the agents. Additional experimental results are provided in our appendices.

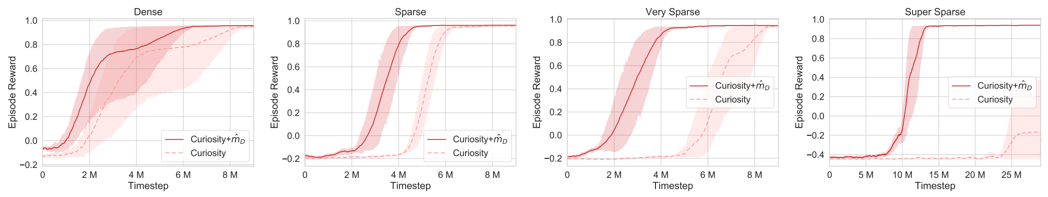
4.5 Transferability of the Constructed Macros to Harder Environments

In order to investigate the transferability of the constructed macros among similar environments, we first execute Algorithm 1 with A2C and ICM (together denoted as "*Curiosity*") on *Dense* to construct a macro $\hat{m}_{\text{D}}$. We choose to train on *Dense* because the agent can learn from both the dense and sparse reward scenarios at different spawn points, enabling it to adapt to different reward sparsity settings easier. We then form the augmented action space $\mathcal{M} = A \cup \{\hat{m}_{\text{D}}\}$ and train the agent to use $\mathcal{M}$ with *Curiosity* on similar but harder environments with sparser rewards. The results are plotted in Fig. 4. Figs. 4 (a)∼4 (d) demonstrate that the agents with $\mathcal{M}$ learn significantly faster than the agents without it. These results



(a) *BeamRider*. (b) *Q*bert*. (c) *Pong*. (d) *Breakout*.

Figure 3: (a) Learning curves of the baseline DRL methods with/without the uses of the constructed macros. (b)(c)(d) Learning curves of the baseline DRL methods with/without the uses of $\hat{m}_{\text{A2C}}$.



(a) *Dense*. (b) *Sparse*. (c) *Very Sparse*. (d) *Super Sparse*.

Figure 4: Learning curves of *Curiosity* with/without $\hat{m}_{\text{D}}$ for four different *ViZDoom* tasks.

thus validate the transferability of the constructed macros to harder environments. Fig. 4 also reveals that the gap between '*Curiosity*+$\hat{m}_{\text{D}}$' and '*Curiosity*' grows as the sparsity of the reward signal increases. For the *Super Sparse* task, '*Curiosity*+$\hat{m}_{\text{D}}$' converges at around 13M timesteps, while '*Curiosity*' just begins to learn at about 25M timestep. We compare mean time required to reach the goal w/ and w/o $\hat{m}_{\text{D}}$ over 100 episodes in our appendices.

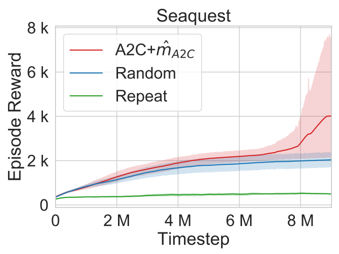
The macro action $\hat{m}_{\text{D}}$ constructed by Algorithm 1 is $(2, 2, 1)$, which corresponds to two forward moves followed by one right turn. We speculate that this macro action is transferable because the map layouts favor forward moves and right turns. Moving forward is more essential than turning right for the four *ViZDoom* tasks, as the optimal trajectories comprise of a number of straight moves. On the contrary, left turn is rarely performed by the agent, as this choice of action is only adopted when the agent is deviated from its optimal routes.

4.6 Ablation Analysis

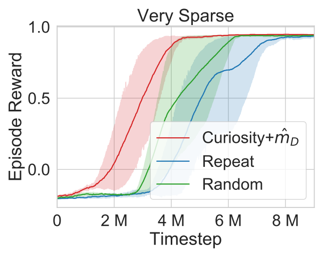
To justify our decision of selecting GA as our macro construction method, we additionally compare GA with three different macro construction methods to validate that GA is superior to them in most cases and possess several promising properties required by our methodology.

Comparison with random macros. We first compare the performance of the macros constructed by Algorithm 1 against randomly constructed macros. We present the learning curves of 'A2C with $\hat{m}_{\text{A2C}}$' and 'A2C with $\hat{m}_{\text{D}}$' against 'A2C with the best random macro $\hat{m}_{\text{Random}}$' in Figs. 5 (a) and 5 (b), respectively. For a fair comparison, we randomly select 50 out of all possible macros with lengths from 2 to 5, and evaluate each random macro for 5M timesteps for 2 times similar to how GA evaluates. We limit the lengths of the macros from 2 to 5 because it is the range GA has attempted. Following this procedure, the macro with highest evaluation score is selected to be $\hat{m}_{\text{Random}}$. The agents are then trained for 10M timesteps, and the results are plotted in Figs. 5 (a) and 5 (b) (denoted as "*Random*"). It is observed that the two curves for 'A2C with $\hat{m}_{\text{Random}}$' are not obviously superior to those of 'A2C with $\hat{m}_{\text{A2C}}$' and 'A2C with $\hat{m}_{\text{D}}$'. We further compare the 50 randomly constructed macros with the 50 macros constructed in the course of Algorithm 1, and summarize their impacts on performance as distributions over mean scores in Fig. 5 (c). It is observed that a larger portion of the macros constructed by our approach tend to result in higher mean scores than those constructed randomly. A breakdown and analysis are offered in our appendices.

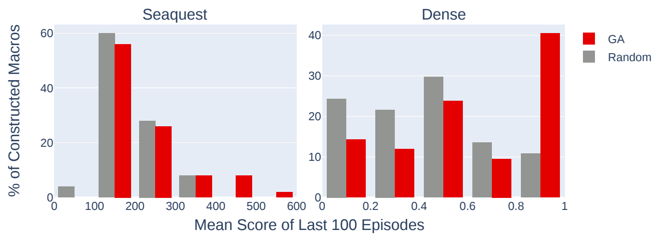
Comparison with action repeat. We next compare the performance of 'A2C with $\hat{m}_{\text{A2C}}$' and 'A2C with $\hat{m}_{\text{D}}$' against 'A2C with the best action repeat macro $\hat{m}_{\text{Repeat}}$', and plot the learning curves in Figs. 5 (a) and 5 (b). In order to construct $\hat{m}_{\text{Repeat}}$ for the two figures,



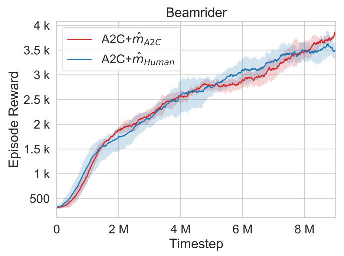
(a) *Seaquest*.



(b) *Very Sparse*.



(c) *Seaquest* and *Dense*.



(d) *BeamRider*.

Figure 5: (a) We compare our methodology to random macros by averaging the episode rewards of the top 10% performing macros. For action repeat, we pick the best macro from all possible candidates. (b) The best macros constructed by our methodology versus the random macros and action repeat. (c) Distribution of the constructed macros over different ranges of the evaluated mean scores. (d) Comparison of the macros constructed from random and handcrafted initial populations.

we evaluate all possible action repeat macros with the same length to $\hat{m}_{\text{A2C}}$ and $\hat{m}_{\text{D}}$ for five random seeds and for 10M timesteps. The macro with highest evaluation score is then selected as $\hat{m}_{\text{Repeat}}$. It is observed that the curves of 'A2C with $\hat{m}_{\text{Repeat}}$' are significantly worse than those of 'A2C with $\hat{m}_{\text{A2C}}$', 'A2C with $\hat{m}_{\text{D}}$', and 'A2C with $\hat{m}_{\text{Random}}$' in the figures. The low performance of them is probably due to the lack of diversity in $\hat{m}_{\text{Repeat}}$.

Comparison with handcrafted macros. We further investigate whether human intervention affects the outcome of Algorithm 1. As the initial population of the macros used by Algorithm 1 are randomly constructed, in this ablation study we replace the initial population by handcrafted macros $(1, 1, 1), (1, 1, 1, 1), (1, 1, 1, 1, 1)$, which correspond to consecutive 'fire' action in *BeamRider*. These handcrafted macros are advantageous to the agent from the perspective of human players, as waves of enemies continue to emerge throughout the game. The macro constructed in this manner is denoted as $\hat{m}_{\text{Human}}$. Fig. 5 (d) shows a comparison of the learning curves of 'A2C with $\hat{m}_{\text{A2C}}$' and 'A2C with $\hat{m}_{\text{Human}}$'. It is observed that the two curves are almost aligned. From the analysis, we therefore conclude that our approach does not need human intervention, and is unaffected by the initial population of the macros.

5 Conclusions

We have formally presented a methodology to construct macro actions that may potentially improve both the performance and learning efficiency of the existing DRL methods. The methodology falls within the scope of a broader framework that permits other possible combinations of the DRL method, the action space augmentation method, the evaluation method, as well as the macro action construction method. We formulated the proposed methodology as a set of algorithms, and used them as the basis for investigating the interesting properties of macro actions. Our results revealed that the macro actions constructed by our methodology are complementary to two representative DRL methods, and may demonstrate transferability among different DRL methods and similar environments. We additionally compared our methodology against three other macro construction methods to justify our design decisions. Our work paves a way for future research on macros and their applications.

References

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with deep reinforcement learning. *arXiv:1312.5602*, Dec. 2013.

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):529–533, Feb. 2015.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proc. Int. Conf. Machine Learning (ICML)*, pages 1928–1937, Jun. 2016.

T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv:1703.03864*, Sep. 2017.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, Aug. 2017.

D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette. Evolutionary algorithms for reinforcement learning. *J. Artificial Intelligence Research (JAIR)*, 11:241–276, 1999.

F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv:1712.06567*, Apr. 2018.

R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. VIME: Variational information maximizing exploration. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pages 1109–1117, Dec. 2016.

R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, Aug. 1999.

A. Vezhnevets, V. Mnih, S. Osindero, A. Graves, O. Vinyals, J. Agapiou, et al. Strategic attentive writer for learning macro-actions. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pages 3486–3494, Dec. 2016.

T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pages 3675–3683, Dec. 2016.

P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proc. the Thirty-First AAAI Conf. Artificial Intelligence (AAAI-17)*, pages 1726–1734, Feb. 2017.

K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman. Meta learning shared hierarchies. In *Proc. Int. Conf. Learning Represenations (ICLR)*, Apr.-May 2017.

C. Daniel, H. Van Hoof, J. Peters, and G. Neumann. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104(2-3):337–357, Sep. 2016.

C. Florensa, Y. Duan, and P. Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *Proc. Int. Conf. Learning Represenations (ICLR)*, Apr. 2017.

M. C. Machado, M. G. Bellemare, and M. Bowling. A laplacian framework for option discovery in reinforcement learning. In *Proc. of the 34th International Conf. on Machine Learning (ICML)*, volume 70, pages 2295–2304, Aug. 2017.

S. Girgin, F. Polat, and R. Alhajj. Improving reinforcement learning by using sequence trees. *Machine Learning*, 81(3):283–331, Dec. 2010.

N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, and D. Silver. Learning and transfer of modulated locomotor controllers. *arXiv:1610.05182*, Oct. 2016.

R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, Sep. 1971.

L. Siklossy and C. Dowson. The role of preprocessing in problem solving systems. In *Proc. Int. Joint Conf. Artificial Intelligence (IJCAI)*, pages 465–471, Aug. 1977.

S. Minton. Selectively generalizing plans for problem-solving. In *Proc. Int. Joint Conf. Artificial Intelligence (IJCAI)*, pages 596–599, Aug. 1985.

M. Pickett and A. G. Barto. PolicyBlocks: An algorithm for creating useful macro-actions in reinforcement learning. In *Proc. Int. Conf. Machine Learning (ICML)*, pages 506–513, Jul. 2002.

A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer. Macro-FF: Improving AI planning with automatically learned macro-operators. *J. Artificial Intelligence Research (JAIR)*, 24:581–621, Oct. 2005.

M. Newton, J. Levine, and M. Fox. Genetically evolved macro-actions in AI planning problems. *Proc. the UK Planning and Scheduling Special Interest Group (PlanSIG) Wksp.*, pages 163–172, 2005.

M. A. H. Newton, J. Levine, M. Fox, and D. Long. Learning macro-actions for arbitrary planners and domains. In *Proc. Int. Conf. Automated Planning and Scheduling (ICAPS)*, pages 256–263, Sep. 2007.

J. J. DiStefano III, A. R. Stubberud, and I. J. Williams. *Feedback and control systems*. McGraw-Hill, 1967.

I. P. Durugkar, C. Rosenbaum, S. Dernbach, and S. Mahadevan. Deep reinforcement learning with macro-actions. *arXiv:1606.04615*, Jun. 2016.

J. Randlov. Learning macro-actions in reinforcement learning. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pages 1045–1051, Dec. 1999.

T. Yoshikawa and M. Kurihara. An acquiring method of macro-actions in reinforcement learning. In *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, pages 4813–4817, Nov. 2006.

H. Onda and S. Ozawa. A reinforcement learning model using macro-actions in multi-task grid-world problems. In *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, pages 3088–3093, Oct. 2009.

F. M. Garcia, B. C. da Silva, and P. S. Thomas. A compression-inspired framework for macro discovery. In *Proc. of the 18th International Conf. on Autonomous Agents and Multi Agent Systems (AAMAS)*, pages 1973–1975, May. 2019.

A. S. Lakshminarayanan, S. Sharma, and B. Ravindran. Dynamic action repetition for deep reinforcement learning. In *Proc. the Thirty-First AAAI Conf. Artificial Intelligence (AAAI-17)*, Feb. 2017.

S. Sharma, A. S. Lakshminarayanan, and B. Ravindran. Learning to repeat: Fine grained action repetition for deep reinforcement learning. In *Proc. Int. Conf. Learning Represenations (ICLR)*, Apr.-May 2017.

A. McGovern, R. S. Sutton, and A. H. Fagg. Roles of macro-actions in accelerating reinforcement learning. In *Proc. Grace Hopper celebration of women in computing*, volume 1317, 1997.

K. Heecheol, M. Yamada, K. Miyoshi, and H. Yamakawa. Macro action reinforcement learning with sequence disentanglement using variational autoencoder. *arXiv:1903.09366*, May 2019.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, 2018.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv:1606.01540*, Jun. 2016.

M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowskim. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conf. Computational Intelligence and Games (CIG)*, pages 1–8, Sep. 2016.

J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 1979.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artificial Intelligence Research (JAIR)*, 47: 253–279, Jun. 2013.

M. Wydmuch, M. Kempka, and W. Jaśkowski. ViZDoom competitions: Playing Doom from pixels. *IEEE Trans. Games*, 2018.

D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proc. Int. Conf. Machine Learning (ICML)*, Aug. 2017.

A. Hill, A. Raffin, M. Ernestus, A. Gleave, R. Traore, et al. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

Appendices

The appendices are organized as follows. Section A1 lists our essential notations. Section A2 describes the implementations of the genetic operators. Section A3 summarizes the hyperparameters used. Section A4 shows the additional experimental results. Finally, Section A5 describes our computing infrastructure.

A1 Notations

Table A1: Essential notations.

Symbol	Name	Symbol	Name
$\lvert\cdot\rvert$	sequence length or set size	$\mathcal{E}$	environment
$\mathbb{P}\{\cdot\}$	probability	$\mathcal{R}$	DRL method
$\mathbb{E}\{\cdot\}$	expected value	$\mathfrak{M}$	macro action space
t	timestep	$\mathcal{M}$	augmented action space
$\mathcal{S}$	state space	m	primitive action or macro action
s	regular state	m_t	m taken at t
s_t	s perceived by agent at t	r_s^m	(discounted) expected reward
$\mathcal{A}$	primitive action space	$\nu(s, m)$	policy over $\mathcal{M}$
a	primitive action	$p_{ss'}^m$	environment dynamic
r_t	one-step reward	$V^\nu(s)$	value function
γ	discount factor	$V_{\mathcal{M}}^*(s)$	optimal value function
$\mathscr{R}$	DRL method collection	$\mathscr{E}$	evaluation method collection
$\mathscr{A}$	augmentation method collection	$\mathscr{C}$	macro construction method collection

A2 Pseudo Code

A2.1 Constructing Macro Actions

Algorithm A1 Action space augmentation method

```
1: input: Primitive action space $\mathcal{A}$
2: input: Macro action $m$
3: output: Augmented action space $\mathcal{M}$
4: function AUGMENT($\mathcal{A}$, $m$)
5:     return $\mathcal{M} = \mathcal{A} \cup \{m\}$
6: end function
```

Algorithm A2 Macro evaluation method

```
1: input: Environment $\mathcal{E}$; DRL method $\mathcal{R}$; Augmented action space $\mathcal{M}$; Evaluation epochs $N$
2: output: Fitness score of $\mathcal{M}$
3: function EVALUATE($\mathcal{E}$, $\mathcal{R}$, $\mathcal{M}$, $N$)
4:     initialize: $E = [e_1, \ldots, e_N]$ $\forall$ $e \in E, e = 0$
5:     for $i$ in 1 to $N$ do                                    ▷ $N$ epoch
6:         Learn a policy over $\mathcal{M}$ in $\mathcal{E}$ using $\mathcal{R}$
7:         $e_i$ = the average of all "last 100 episode rewards"
8:     end for
9:     return Average of $E$
10: end function
```

A2.2 Genetic Operators

Algorithm A3 Append operator

```
1: input: Primitive action space $\mathcal{A}$
2: input: Macro action $m = (a_1, \ldots, a_{|m|})$
3: output: Mutated macro $m'$
4: function Append($\mathcal{A}$, $m$)
5:     Let $\alpha$ = random element in $\mathcal{A}$
6:     return $m' = (a_1, \ldots, a_{|m|}, \alpha)$
7: end function
```

Algorithm A4 Alteration operator

```
1: input: Primitive action space $\mathcal{A}$
2: input: Macro action $m = (a_1, \ldots, a_{|m|})$
3: output: Mutated macro $m'$
4: function Alter($\mathcal{A}$, $m$)
5:     Let $\alpha$ = random element in $\mathcal{A}$
6:     return $m' = (\alpha, a_2, \ldots, a_{|m|})$
7: end function
```

A3 Implementation Details

We used A2C (Mnih et al., 2016) and PPO (Schulman et al., 2017) implementations from Hill et al. (2018) and established our models on top of them. We implemented ICM (Pathak et al., 2017) along with A2C (together denoted as "*Curiosity*") instead of Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016), and found that our implementations tend to be stabler in terms of the variance of learning curves from several runs. The detailed hyperparameters for each method are summarized in Table A2. The GA-related parameters used in Algorithm 1 are presented in Table A3. The primitive actions we used are summarized in Table A4.

Table A2: List of the hyperparameters for the baseline methods.

Hyperparameter	A2C	PPO	Curiosity
Discount factor	0.99	0.99	0.99
Number of frame skip	4	4	4
Number of parallel environments	16	16	20
Rollout length	5	128	20
Batch size	80	2048	400
Value function coefficient	0.25	0.5	0.5
Entropy coefficient	0.01	0.01	0.01
Gradient clipping maximum	4.0	0.5	40.0
Learning rate	0.001	0.0003	0.0003
Learning rate annealing	Linear		Constant
Clipping parameter		0.2	
Policy gradient loss weight λ			0.1
Forward model loss β			0.2
Intrinsic reward scaling factor			0.01

Table A3: List of the GA-related parameters used in Algorithm 1.

Parameter	q	q_+	q_*
Value	8	5	3

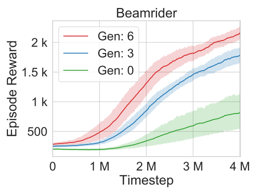
Table A4: List of the primitive actions.

Environment		0	1	2	3	4	5
	BeamRider	`NOOP`	`FIRE`	`RIGHT`	`LEFT`		
	Breakout	`NOOP`	`FIRE`	`RIGHT`	`LEFT`		
	Enduro	`NOOP`	`FIRE`	`RIGHT`	`LEFT`	`DOWN`	
Atari	*Pong*	`NOOP`	`RIGHT`	`LEFT`			
	*Q*bert*	`NOOP`	`UP`	`RIGHT`	`LEFT`	`DOWN`	
	Seaquest	`NOOP`	`FIRE`	`UP`	`RIGHT`	`LEFT`	`DOWN`
	SpaceInvaders	`NOOP`	`FIRE`	`RIGHT`	`LEFT`		
ViZDoom		`TURN_LEFT`	`TURN_RIGHT`	`MOVE_FORWARD`			

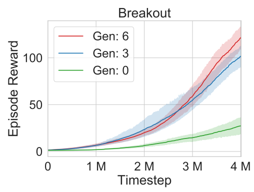
A4 Experimental Results

A4.1 Additional Comparison of the Learning Curves

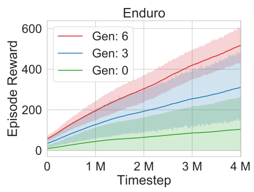
Figs. A1, A2 and A3 present results unable to be completely included in Sections 4.2, 4.3 and 4.4, respectively.



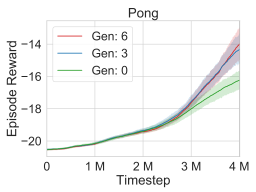
(a) *BeamRider*.



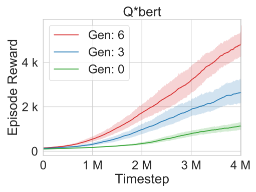
(b) *Breakout*.



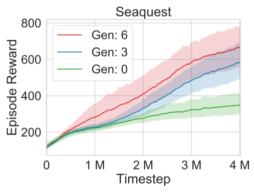
(c) *Enduro*.



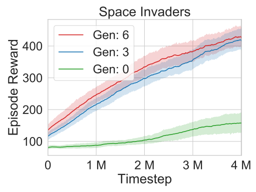
(d) *Pong*.



(e) *Q*bert*.

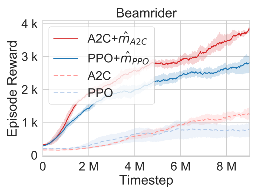


(f) *Seaquest*.

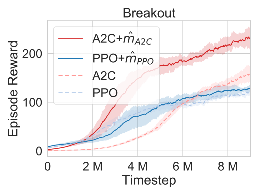


(g) *SpaceInvaders*.

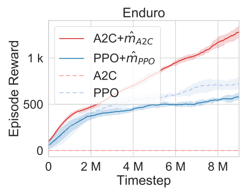
Figure A1: Learning curves of the A2C agents using different generations of the constructed macros. Please note that "Gen" denotes "generation". Each curve along with the corresponding confidence interval stands for the average performance of the best macros in the generation (i.e., Q in Algorithm 1).



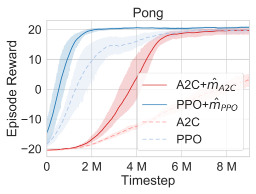
(a) *BeamRider*.



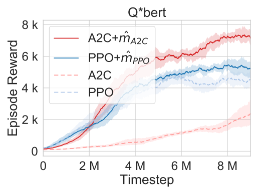
(b) *Breakout*.



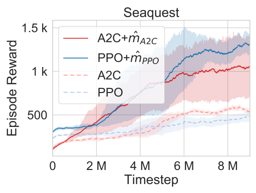
(c) *Enduro*.



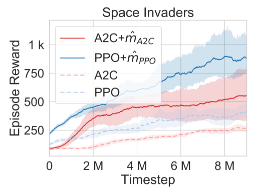
(d) *Pong*.



(e) *Q*bert*.

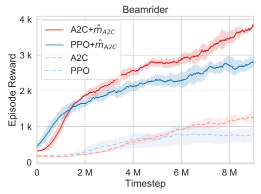


(f) *Seaquest*.

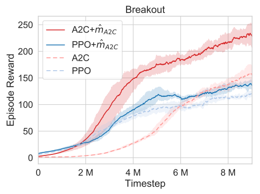


(g) *SpaceInvaders*.

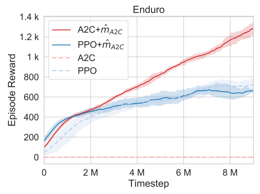
Figure A2: Learning curves of the baseline methods with/without the constructed macros for evaluating compatibility.



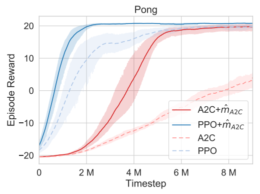
(a) *BeamRider*.



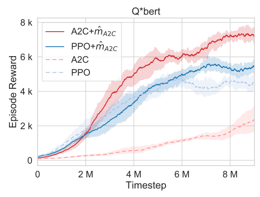
(b) *Breakout*.



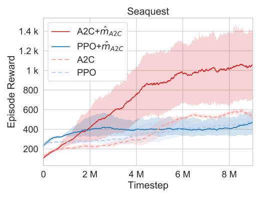
(c) *Enduro*.



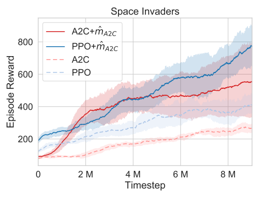
(d) *Pong*.



(e) *Q*bert*.



(f) *Seaquest*.



(g) *SpaceInvaders*.

Figure A3: Learning curves of the baseline methods with/without $\hat{m}_{\mathrm{A2C}}$ for evaluating transferability.

A4.2 Additional Evaluation Results

Table A5 presents additional promising results complementary to Section 4.5.

Table A5: Evaluation results of the four *ViZDoom* tasks. We present the mean timesteps required to reach the goal over 100 episodes. Please note that the constructed macro $\hat{m}_{\mathrm{D}} = (2, 2, 1)$.

Environment	Curiosity	Curiosity+$\hat{m}_{\mathrm{D}}$	**Reduction**
Dense	120.43	94.52	**-21.51%**
Sparse	157.46	139.69	**-11.29%**
Very Sparse	237.45	161.82	**-31.85%**
Super Sparse	405.63	223.95	**-44.79%**

A4.3 Detailed Comparison of the Macros Constructed by Algorithm 1 with Random Macros

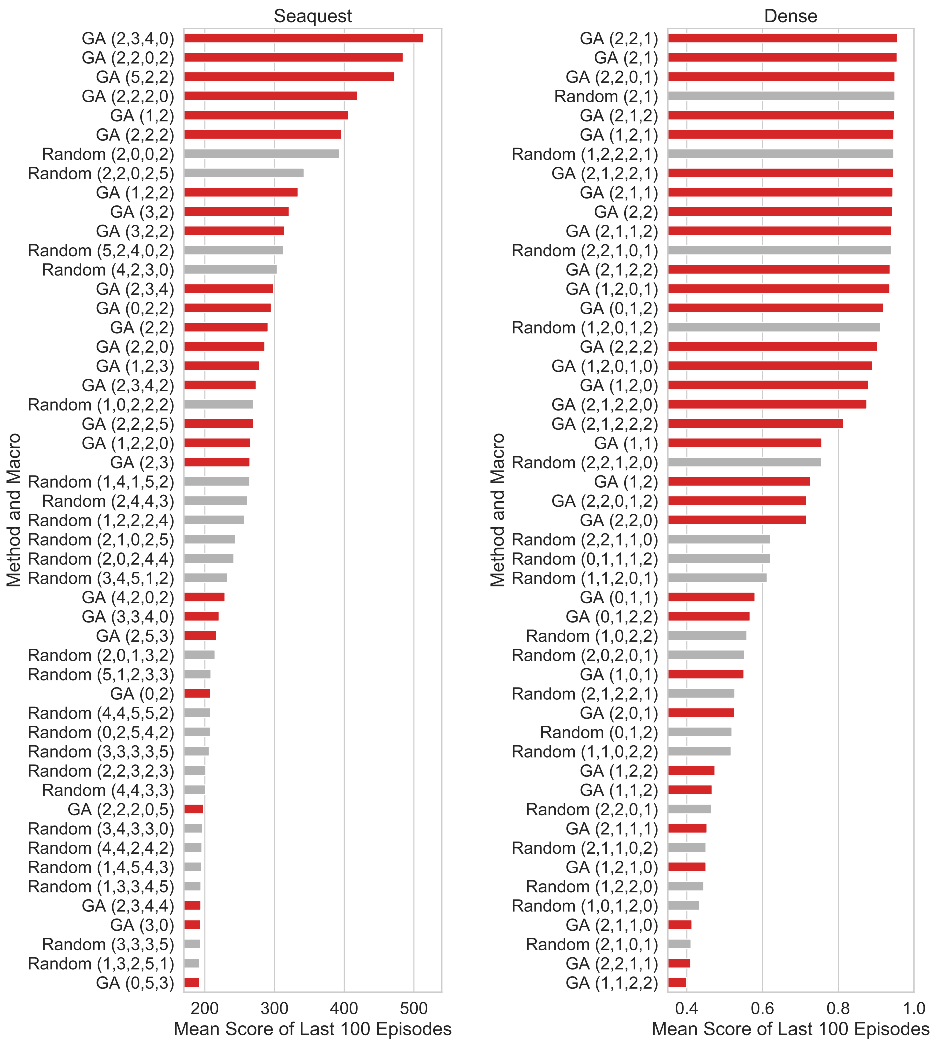
In addition to the results presented in Section 4.6, we further illustrate a more detailed comparison of the macros constructed by our methodology and the randomly constructed macros in Fig. A4.

A5 Computing Infrastructure

In this study, we use both our customized machines and the virtual machines named *n1-statnd-64* on Google Cloud Platform (GCP). We list the specification per instance in Table A6. In total, we utilized up to 320 virtual CPU cores, 8 graphics cards, and around 1 TB memory in our experiments.

Table A6: Specification of our computing infrastructure.

Component	Our Customized Machine	n1-statnd-64
Processor	AMD Ryzen™ Threadripper™ 2990WX (32C/64T)	32 vCPUs
Memory	128 GB	120 GB
Hard Disk Drive	6 TB	100 GB
Solid-State Disk	1 TB NVMe PCIe Internal SSD	
Graphics Card	NVIDIA GeForce® RTX 2080 Ti (4 per instance)	



(a) *Atari Seaquest*. (b) *ViZDoom Dense*.

Figure A4: Performance comparison of the macros constructed via Algorithm 1 (the red bars) and the random construction method described in Section 4.6 (the gray bars) in terms of the mean scores of the last 100 episodes achieved by the A2C agents using the macros. Please note that we present only the results of the top 50 performers out of the constructed macros for the *Seaquest* and *Dense* tasks.