A Compression-Inspired Framework for Macro Discovery

Extended Abstract

Francisco M. Garcia
University of Massachusetts
Amherst, Massachusetts, USA
fmgarcia@cs.umass.edu

Bruno C. da Silva
Federal University of
Rio Grande do Sul
Porto Alegre, Rio Grande, Brazil
bsilva@inf.ufrgs.br

Philip S. Thomas
University of Massachusetts
Amherst, Massachusetts, Brazil
pthomas@cs.umass.edu

ABSTRACT

We consider the problem of how a reinforcement learning agent, tasked with solving a *set* of related Markov decision processes, can use knowledge acquired early on in its lifetime to improve its ability to more rapidly solve novel tasks. We propose a three-step framework that generates a diverse set of macros that lead to high rewards when solving a set of related tasks. Our experiments show that augmenting the original action-set of the agent with the identified macros allows it to more rapidly learn optimal policies in novel MDPs.

KEYWORDS

Reinforcement Learning; Hierarchical RL; Exploration

ACM Reference Format:

Francisco M. Garcia, Bruno C. da Silva, and Philip S. Thomas. 2019. A Compression-Inspired Framework for Macro Discovery. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 3 pages.

1 INTRODUCTION

One of the key aspects of human learning is our ability to construct building blocks upon which we can learn new skills. Humans generally bootstrap higher-level skills acquired early on in their lives to solve new problems. For example, a child learning to run does not need to re-learn the "balance" skill to stand in two feet as part of learning a "run" skill. Instead, the child uses previously acquired skills to more efficiently explore the consequences of his actions when facing novel tasks. In the Reinforcement Learning (RL) literature, higher-level actions are sometimes called *options* or *macros* [6, 8]. They introduce a bias in the behavior of the agent which is key during exploration to more efficiently learn how to solve novel problems. Formally, a macro of length l is a sequence of actions $m = (a_1, \cdots, a_l)$, where $m_{(i)}$ denotes the i^{th} action in macro m. If carefully constructed, macros have been shown to improve learning speed by allowing an agent to quickly reach distant areas of the state space during training [4].

We assume that if two tasks (MDPs) are related (i.e., belong to a same class) then the behavior learned as a solution to the former task can be used to identify macros that will help in learning how to solve the latter. We define a *problem class* C to be the set of all related tasks, c, that an agent may face, where $c = (\mathcal{S}, \mathcal{A}, P_c, R_c, \gamma, d_0^c)$. That is, each task, c, in the same problem class corresponds to a Markov decision process defined by the same state space $\mathcal{S}$, action space $\mathcal{A}$ and discount factor γ, but with a task-specific transition function P_c, reward function R_c, and initial state distribution d_0^c. The question this paper focuses on is: *"How can an agent identify and leverage useful macros for a given class or distribution of problems?"*.

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

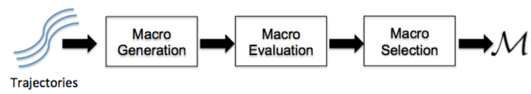


Figure 1: Diagram depicting proposed framework.

2 PROBLEM STATEMENT

We consider the setting where an agent is required to solve a set of tasks $c \in C$, where C is a given problem class, and assume that when solving a particular task, it can interact with it for I episodes. After the agent has trained on a subset $C_{train} \subset C$ of tasks, we are interested in identifying a set of macros to be used for improving learning in the set of remaining tasks $C_{test} \subset C$.

We define the performance of a set of macros $\mathcal{M}$ in a particular task c to be $\rho(\mathcal{M}, c) = \mathbf{E}\left[\frac{1}{I}\sum_{i=1}^{I}\sum_{t=0}^{T}\gamma^t R_t^i \middle| \mathcal{A}_{\mathcal{M}}, c\right]$, where R_t^i is the reward at time step t during the i^{th} episode. This quantity expresses the expected average return an agent gets over I episodes on a task c using an extended action set $\mathcal{A}_{\mathcal{M}} = \mathcal{A} \cup \mathcal{M}$ (an action set composed of primitives and macros). In other words, the performance of a set of macros is defined by how much the performance of an agent is improved during training by augmenting the agent's original action set with a given set of macros $\mathcal{M}$. Our goal is to find one (of possibly many) optimal set of macros $\mathcal{M}^*$ for C' such that $\mathcal{M}^* \in \arg\max_{\mathcal{M}}\left(\frac{1}{|C_{test}|}\sum_{c \in C_{test}} \rho(\mathcal{M}, c)\right)$. Unfortunately, the domain of this objective function is discrete, making it non-differentiable and difficult to optimize. We posit that *compression techniques* provide a means to identify macros representing recurring behaviors in optimal policies for problems in the class, which therefore are good candidates for composing $\mathcal{M}^*$. Full details and derivations of this framework can be found at https://arxiv.org/abs/1711.09048.

3 A HEURISTIC FOR APPROXIMATING $\mathcal{M}^*$

The proposed framework is summarized in the diagram in Figure 1. The agent first trains on a set of tasks $C_{train} \subset C'$, thereby acquiring an optimal policy π_c^* for each task. It then samples n trajectories from each policy π_c^* for task c. Once these samples have been obtained, our framework generates a set of macros $\mathcal{M}'$ as an approximation to $\mathcal{M}^*$ via the following 3-step process:

Macro Generation: A trajectory τ of length l is the sequence of states, actions and rewards experienced by an agent following a given policy: $\tau = \{s_0, a_0, r_0, \ldots, s_{l-1}, a_{l-1}, r_{l-1}\}$. We define an action-trajectory $\tau_a = \{a_0, a_1, \ldots, a_{l-1}\}$ as the sequence of actions in τ. Our agent is first trained on a set of training tasks and generates a set of action-trajectories by sampling them from the learned optimal policies. To generate macros, we consider each action-trajectory akin to a message we wish to compress and the set of primitive actions, $\mathcal{A}$, analogous to the symbols in the initial alphabet used for compression. In this work we use the LZW compression algorithm [10]. Algorithm 1 shows our adaptation to encode action-trajectories as macros.

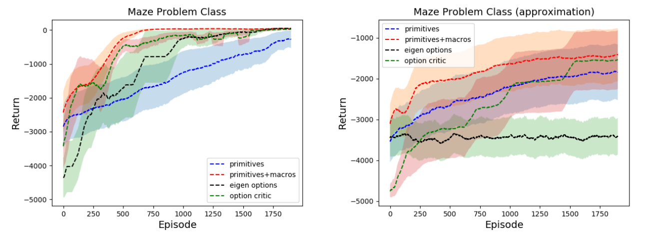
Algorithm 1 LZW - macro codebook generation

```
Initialize alphabet Σ = 𝒜
macro m = ()
for each action-trajectory τ_a do
    for each action a in τ_a do
        m = m + a
        if m ∉ Σ then
            Σ = Σ ∪ {m}
            m = ()
return Σ
```

Macro Evaluation: At this stage we have generated a possibly large set of candidate macros, $\mathcal{M}$, but we do not know how useful they are in helping the agent to quickly solve new tasks. To do this, we measure the utility of a macro m w.r.t C by means of a *U-function* defined as $U_C^\pi(m) = \mathbf{E}\left[Q_C^\pi(S, m)\right]$, where the expectation is over tasks in C and states S sampled from the *on-policy distribution* [7]. In other words, the utility of a macro m is the expected Q-value of m over all states in the problem class. A benefit of this approach is that given the Q-values of primitives, the Q-value of a *macro* can be computed efficiently, in closed-form, as:

$$
\begin{aligned}
Q_c^\pi(s, m) = \sum_{k=0}^{l_m - 1} \Bigg(& \gamma^k \Big[\sum_{s^{(1)} \in \mathcal{S}} \cdots \sum_{s^{(l_m)} \in \mathcal{S}} \frac{\Pi_{i=0}^{l_m - 1} P_c(s^i, m_{(i)}, s^{i+1})}{P_c(s^{(k)}, m_{(k)}, s^{(k+1)})} \Big] \\
& \times \Big[Q(s^k, m_{(k)}) - \sum_{s^{k+1} \in \mathcal{S}} P_c(s^{(k)}, m_{(k)}, s^{(k+1)}) \\
& \times \gamma \sum_{a' \in \mathcal{A}} \pi(a', s^{(k+1)}) Q(s^{(k+1)}, a') \Big] \Bigg) \\
& + \sum_{s^{(1)} \in \mathcal{S}} \cdots \sum_{s^{(l_m)} \in \mathcal{S}} \Pi_{i=0}^{l_m - 1} P_c(s^{(i)}, m_{(i)}, s^{(i+1)}) \\
& \times \gamma \sum_{a' \in \mathcal{A}} \pi(a', s^{(l_m)}) Q(s^{(l_m)}, a').
\end{aligned}
$$

Macro Selection: The utility allows us to asses which macros may lead to higher rewards. However, we must also consider that if too many macros are added to $\mathcal{A}_\mathcal{M}$, the agent's action-set may get too large, thereby hindering learning. To address this issue, we select macros that not only have a high utility, but that are diverse from each other. Let S_t be a random variable denoting the state where m is executed and S_{t+l_m} the state where m finishes execution. Furthermore, let $S' = d(S_{t+l_m}, S_t)$ be a random variable denoting the change in state from executing a macro m, where d is a function measuring the change in state, and let p_m be a probability distribution over S' for m. For two macros m_1 and m_2, we define the distance between them to be the KL divergence between p_{m_1} and p_{m_2}: $D_{KL}(p_{m_1} || p_{m_2}) = -\sum_{S'} p_{m_1}(S') \log\left(\frac{p_{m_2}(S')}{p_{m_1}(S')}\right)$. The set $\mathcal{M}'$ is incrementally built by only including those macros (obtained from Algorithm 1) that have a minimum distance δ to all other macros already included in the set. By selecting macros in descending order according to their U-value, the utility defines a preference criterion by which macros are selected. Pseudocode for framework is given in Algorithm 2.



(a) Average performance in maze navigation when transition graph is kept fixed throughout all training and testing tasks.

(b) Average performance in maze navigation when transition graphs vary between training and testing.

Figure 2: Average performance comparison on 20 novel test tasks. Start and goal locations were selected randomly.

Algorithm 2 Macro discovery process

```
1. Macro Generation
Learn optimal policy π*_c for all c ∈ C_train.
Collect action-trajectories τ_a from each π*_c in task c.
Generate macros 𝓜 from all τ_a by Algorithm 1
2. Macro Evaluation
Sort all m ∈ 𝓜 by U_C^{π*_c}(m) in descending order.
3. Macro Selection
𝒜_𝓜' = 𝒜
for m ∈ 𝓜 do
    if min D_KL(p_m || p_m') > δ, ∀m' ∈ 𝒜_𝓜' then
        𝒜_𝓜' = 𝒜_𝓜' ∪ {m'}
```

4 EXPERIMENTAL RESULTS

We compared the performance of primitives (P), primitive+macros (M), eigenoptions (E) [3] and the option-critic architecture (O) [1]. We evaluated our method in a maze navigation problem class where the environment dynamics are known andthe true Q-values (for primitives) can be estimated accurately in tabular form. This allows us to accurately compute the U-value for any candidate macro. Figure 2 compares the learning performance of an agent using each technique with tabular Q-learning as the learning algorithm. We also extended these test to two problems with large state spaces, where the transition functions and U-values had to be estimated by sampling: **Animat** [9] and **Lunar Lander** [2]. We tested our identified macros on 10 novel testing tasks and used DQN Mnih et al. [5] as a learning algorithm. The return of the policy learned after 1000 training episodes for the two problem classes is as follows. In **Animat**, P: -909.77 ± 199.53, M: $\mathbf{-752.89 \pm 188.59}$, E: $-1,432.46 \pm 64.72$, O: $1,955.47 \pm 41.22$. In **Lunar Lander**, P: -314.03 ± 44.09, M: $\mathbf{-246.89 \pm 28.99}$, E: -266.43 ± 5.22, O: -265.51 ± 7.42

ACKNOWLEDGMENTS

This work was partially supported by FAPERGS under grant no. 17/2551-000.

REFERENCES

[1] Pierre-Luc Bacon, Jean Harb, and Doina Precup. 2017. The Option-Critic Architecture. In *AAAI*.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. (2016). http://arxiv.org/abs/1606.01540 cite arxiv:1606.01540.

[3] Michael Bowling Marlos C. Machado, Marc G. Bellemare. 2017. A Laplacian Framework for Option Discovery in Reinforcement Learning. *CoRR* (2017).

[4] A. McGovern and R. Sutton. 1998. *Macro Actions in Reinforcement Learning: An Empirical Analysis*. Technical Report. University of Massachusetts - Amherst, Massachusetts, USA.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (Feb. 2015), 529–533. http://dx.doi.org/10.1038/nature14236

[6] Jette Randl. 1998. Learning Macro-Actions in Reinforcement Learning. In *NIPS*.

[7] Richard S. Sutton and Andrew G. Barto. 2018. *Introduction to Reinforcement Learning* (2nd ed.). MIT Press, Cambridge, MA, USA.

[8] Richard S. Sutton, Doina Precup, and Satinder P. Singh. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence* 112, 1-2 (1999), 181–211.

[9] Philip S. Thomas and Andrew G. Barto. 2011. Conjugate Markov Decision Processes. In *Proceedings of the 28th International Conference on International Conference on Machine Learning (ICML'11)*. Omnipress, USA, 137–144. http://dl.acm.org/citation.cfm?id=3104482.3104500

[10] T. A. Welch. 1984. A Technique for High-Performance Data Compression. *Computer* 17, 6 (June 1984), 8–19. https://doi.org/10.1109/MC.1984.1659158