Hierarchical Reinforcement Learning with Unlimited Recursive Subroutine Calls

Yuuji Ichisugi[1](y-ichisugi@aist.go.jp)*, Naoto Takahashi[1], Hidemoto Nakada[1], and Takashi Sano[2]

[1] National Institute of Advanced Industrial Science and Technology (AIST), AIRC
[2] Department of Computer and Information Science, Faculty of Science and Technology, Seikei University

Abstract. Humans can set suitable subgoals to achieve certain tasks. They can also set sub-subgoals recursively if required. The depth of this recursion is apparently unlimited. Inspired by this behavior, we propose a new hierarchical reinforcement learning architecture called RGoal. RGoal solves the Markov Decision Process (MDP) in an augmented state-action space. In multitask settings, sharing subroutines between tasks makes learning faster. A novel mechanism called thought-mode is a type of model-based reinforcement learning. It combines learned simple tasks to solve unknown complicated tasks rapidly, sometimes in zero-shot time.

Keywords: Hierarchical reinforcement learning, Model-based reinforcement learning, Zero-shot learning, Computational neuroscience

1 Introduction

Humans can set suitable subgoals to achieve certain tasks (goals). They can also set sub-subgoals recursively if needed. For example, if you wish to get an object on a high shelf, it is necessary to set up a ladder first. In this case, "the ladder is set up" is a subgoal state. If the ladder is in a store room, it is necessary to go to the store room first to retrieve the ladder. In this case, "you are in the store room" becomes a sub-subgoal state. The depth of this recursion is apparently unlimited for humans. Inspired by this behavior, we propose a new hierarchical reinforcement learning architecture [2][3][4][5][6][8][9][11] called the *RGoal architecture*.

In RGoal, an agent's subgoal settings are similar to subroutine calls in programming languages. Each subroutine can execute primitive actions or recursively call other subroutines. The timing for calling another subroutine is learned

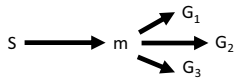

Fig. 1. Even though goals (G_1, G_2, and G_3) are different between tasks, the route from the state S to the subgoal m is common. If the tasks share that route as a subroutine, learning will be accelerated.

Ichisugi Y., Takahashi N., Nakada H., Sano T., Hierarchical Reinforcement Learning with Unlimited Recursive Subroutine Calls. In Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning, Lecture Notes in Computer Science, vol 11728, pp. 103--114, Springer, Cham, 2019.

The final authenticated version is available online at https://doi.org/10.1007/978-3-030-30484-3_9

by using a standard reinforcement learning method. Unlimited recursive subroutine calls accelerate learning because they increase the opportunity for the reuse of subroutines in multitask settings.

RGoal is strongly influenced by the previously proposed HDG[4] and MAXQ[6] architectures. MAXQ is a multi-layered hierarchical reinforcement learning architecture with a fixed number of layers. It accelerates learning based on the following three features.

1. Subtask sharing: In multitask settings, sharing subroutines between tasks makes learning faster (Fig.1).
2. Temporal abstraction: When learning complicated tasks, restricting the search space such that it only includes combinations of simple subroutines makes learning faster.
3. State abstraction: Abstracting states to such an extent that they do not affect the execution of subroutines makes learning faster.

RGoal provides the first feature through *value function decomposition* (Sec.2.3) and the second feature through a novel mechanism called *thought-mode* (Sec.2.7). Although we have not yet implemented the third feature, it should be possible through function approximation using neural networks.

In the future, by extending RGoal, we wish to construct a computational model of the mechanism of human planning based on the prefrontal cortex of the human brain. Therefore, RGoal is designed not only to be useful from an engineering perspective, but also to be a simple architecture that can be easily implemented in the neural circuits of the brain.

The remainder of the paper is organized as follows. First, we describe the architecture of RGoal in Sec.2 and evaluate it in Sec.3. We describe related works in Sec.4. Finally, we present our conclusions in Sec.5.

2 The RGoal architecture

2.1 Landmarks and subgoals

In this paper, we assume that the set of states that may become goals or subgoals on the environment is given beforehand. We refer to an element of this set as a *landmark*. Typically, landmarks are states of the environment that are salient to the agent.

In RGoal, a subroutine g is "a policy for reaching the subgoal state g from arbitrary states." We assume that an agent executing subroutine g will reach the corresponding state g within finite time. This assumption simplifies the theoretical framework and algorithm for RGoal, and facilitates the realization of the thought-mode described in Sec.2.7. It must be possible to extend RGoal in the future such that each subroutine can have more than one terminal state, similar to the MAXQ architecture[6]. Although the reusability of subroutines can be increased in this manner, it may also increase the calculation cost of action-value functions or decrease calculation accuracy.

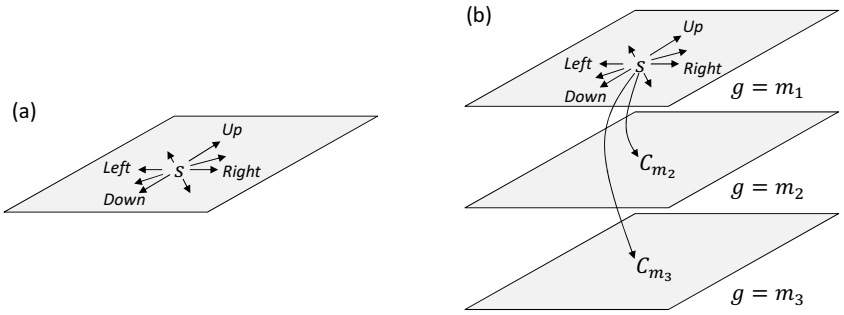

Fig. 2. (a) An example state-action space in a two-dimensional grid. (b) The augmented state-action space when a landmark set $\mathcal{M} = \{m_1, m_2, m_3\}$ is given. Possible actions include the subroutine calls $\mathcal{C}_\mathcal{M} = \{C_{m_1}, C_{m_2}, C_{m_3}\}$ and movements within the two-dimensional space.

An agent maintains a stack to remember subgoals. When an agent calls a subroutine g', the current subgoal g is pushed onto the stack. When the subroutine g' terminates (the agent reaches the corresponding state g'), the original subgoal g is popped from stack and reset as a new subgoal.

There is another design methodology that does not use a stack. In this methodology, an agent only remembers the original goal G. Whenever the final called subroutine terminates, the current subgoal is reset to G. Although we have confirmed that this methodology also works, we do not present its details in this paper.

Although the landmark set affects performance, a bad landmark set does not make a task unsolvable. If a landmark set only contains the goal state, the behavior of RGoal is the same as non-hierarchical reinforcement learning. If there are too many landmarks in the set, learning becomes very difficult. However, landmarks that are not worth using will be gradually filtered out as learning progresses.

2.2 The augmented state-action space

The RGoal architecture learns the action-value function for the *Markov decision process (MDP)* in *the augmented state-action space*[9] described in this Section. Because the mathematical structure of this MDP is the same as typical MDPs, we can utilize various theoretical conclusions (e.g., convergence to an exact solution) and implementation techniques (e.g., function approximation and eligibility trace) to solve problems.

An MDP is defined as $< \mathcal{S}, \mathcal{A}, P, r >$, which consists of a set of states $\mathcal{S}$, set of actions $\mathcal{A}$, transition function $P : \mathcal{S} \times \mathcal{A} \to (\mathcal{S} \to [0, 1])$, and reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. When an MDP and a landmark set $\mathcal{M} = \{m_1, m_2, \cdots\} \subseteq \mathcal{S}$ are given, another MDP on the augmented state-action space $< \tilde{\mathcal{S}}, \tilde{\mathcal{A}}, \tilde{P}, \tilde{r} >$ is defined as follows. First, a set of augmented states $\tilde{\mathcal{S}}$ and a set of augmented

actions $\tilde{\mathcal{A}}$ are defined as

$$\begin{aligned}\tilde{\mathcal{S}} &= \mathcal{S} \times \mathcal{M} \\ \tilde{\mathcal{A}} &= \mathcal{A} \cup \mathcal{C}_{\mathcal{M}}, \quad \mathcal{C}_{\mathcal{M}} = \{C_{m_1}, C_{m_2}, \cdots\}.\end{aligned} \tag{1}$$

An augmented state $\tilde{s} = (s, g) \in \tilde{\mathcal{S}}$ is a pair consisting of an original state s and a subgoal state $g \in \mathcal{M}$. $C_m \in \mathcal{C}_{\mathcal{M}}$ is an action that calls a subroutine m. In other words, C_m sets the landmark m as a new subgoal. Taking an action C_m changes the augmented state (s, g) to (s, m). A transition function $\tilde{P}(\tilde{s}'|\tilde{s}, \tilde{a})$ is defined based on the original transition function $P(s'|s, a)$ as follows:

$$\begin{aligned}\tilde{P}((s', g)|(s, g), a) &= P(s'|s, a) \\ \tilde{P}((s, m)|(s, g), C_m) &= 1.\end{aligned} \tag{2}$$

A reward function $\tilde{r}(\tilde{s}, \tilde{a})$ is defined based on the original reward function $r(s, a)$ as follows:

$$\begin{aligned}\tilde{r}((s, g), a) &= r(s, a) \\ \tilde{r}((s, g), C_m) &= R^{\mathcal{C}},\end{aligned} \tag{3}$$

where the constant $R^{\mathcal{C}}$ is a hyperparameter that represents the cost of each subroutine call.

Fig.2 presents an example of an augmented state-action space. It contains the subgoal g, which represents the agent's inner state, as part of the state of the external environment. If the original state-action space is a two-dimensional space and n landmarks are given, the augmented state-action space looks like a building with n floors. At each step, the agent moves within the current floor or moves to another floor. In general, optimal policies do *not* execute any C_m. However, the execution of some C_m may make convergence to a suboptimal policy faster.

2.3 Value function decomposition

Decomposition of the action-value function in the augmented state-action space makes learning faster because parts of the decomposed functions are shared between different tasks. The details of this process are provided below.

Given a policy $\pi : \tilde{\mathcal{S}} \times \tilde{\mathcal{A}} \to [0, 1]$ and a goal $G \in \mathcal{M}$, the action-value function $Q^{\pi}_{G}(\tilde{s}, \tilde{a})$ is defined as

$$Q^{\pi}_{G}((s, g), \tilde{a}) = E^{\pi}_{G}[\Sigma^{\infty}_{t=0} r_{t+1} | \tilde{s}_0 = (s, g), \tilde{a}_0 = \tilde{a}], \tag{4}$$

which is the expected value of the summation of the sequence of rewards $r_1 = \tilde{r}(\tilde{s}_0, \tilde{a}_0), r_2 = \tilde{r}(\tilde{s}_1, \tilde{a}_1), \cdots$, which are obtained when taking an action $\tilde{a}$ at an initial state $\tilde{s}_0 = (s, g)$ and taking actions according to the policy π. We assume that the total reward obtained after reaching the goal state G is 0. In other words, tasks are episodic. In this paper, we assume that rewards are not discounted.

We assume that if an agent is using a policy π, the agent at a state (s, g) reaches the subgoal state (g, g) within finite time. Furthermore, an agent at the state (g, G) reaches the goal state (G, G) within finite time. When the stack contains only G and an agent reaches the subgoal state (g, g) from (s, g), the state is automatically set to (g, G) and the reward at the time is 0. Then, $Q_G^\pi(\tilde{s}, \tilde{a})$ can be decomposed into two parts (i.e., rewards obtained before and after reaching the subgoal g) as follows:

$$Q_G^\pi((s, g), \tilde{a}) = Q^\pi(s, g, \tilde{a}) + V_G^\pi(g), \tag{5}$$

where $Q^\pi(s, g, \tilde{a})$ is the expected value of the total rewards obtained when taking an action $\tilde{a}$ at an initial state (s, g) and taking actions according to the policy π until reaching (g, g). Additionally, $V_G^\pi(g)$ is the expected value of the total rewards after the state (g, G) until reaching (G, G), which can be efficiently calculated based on $Q^\pi(s, g, \tilde{a})$ as

$$\begin{aligned} V_G^\pi(g) &= \Sigma_{\tilde{a}} \pi((g, G), \tilde{a}) Q_G^\pi((g, G), \tilde{a}) \\ &= \Sigma_{\tilde{a}} \pi((g, G), \tilde{a})(Q^\pi(g, G, \tilde{a}) + V_G^\pi(G)) \\ &= \Sigma_{\tilde{a}} \pi((g, G), \tilde{a}) Q^\pi(g, G, \tilde{a}). \end{aligned} \tag{6}$$

(Note that $V_G^\pi(G) = 0$.)

Because the function $Q^\pi(s, g, \tilde{a})$ does not depend on the original goal G, it can be shared between different tasks to make learning faster. The same argument holds when recursive calls are permitted.

2.4 Update rule

The current implementation of RGoal represents an action-value function $Q(s, g, \tilde{a})$ as a table. The update rule for the table can be derived from a standard reinforcement learning method, Sarsa algorithm for $Q_G(\tilde{s}, \tilde{a})$:

$$Q_G(\tilde{s}, \tilde{a}) \leftarrow Q_G(\tilde{s}, \tilde{a}) + \alpha(r + Q_G(\tilde{s}', \tilde{a}') - Q_G(\tilde{s}, \tilde{a})). \tag{7}$$

Consider the case where $\tilde{a}$ is $C_{g'}$, which represents a subroutine call g'. When the stack contains only G and prior to the subroutine call, the assumed route of the agent is $s \to g \to G$. After the subroutine call, the route is changed to $s \to g' \to g \to G$. Therefore, the following equation holds:

$$\begin{aligned} & Q_G(\tilde{s}', \tilde{a}') - Q_G(\tilde{s}, \tilde{a}) \\ &= (Q(s', g', \tilde{a}') + V_g(g') + V_G(g)) - (Q(s, g, \tilde{a}) + V_G(g)) \\ &= Q(s', g', \tilde{a}') - Q(s, g, \tilde{a}) + V_g(g'). \end{aligned} \tag{8}$$

This equation also holds when $\tilde{a}$ is not a subroutine call, but is a primitive action. (Not that $g = g'$ and $V_g(g') = 0$, in such cases.) The same argument holds when recursive calls are permitted. From Eqs. (5)(7)(8), the update rule for $Q(s, g, \tilde{a})$ is derived as

$$Q(s, g, \tilde{a}) \leftarrow Q(s, g, \tilde{a}) + \alpha(r + Q(s', g', \tilde{a}') - Q(s, g, \tilde{a}) + V_g(g')). \tag{9}$$

Note that special treatment is required when $s = g$ (i.e., the agent reaches the subgoal g). Because $Q(s, g, \tilde{a}) = 0$ when $s = g$, by definition, the table values should not change in such cases.

2.5 Table initialization

The elements of the table Q should be initialized as $Q(s, g, \tilde{a}) = 0$ if $s = g$.

If $s \neq g$, the initial values are arbitrary. However, the values do affect performance[7]. As an extreme case, we can restrict subroutine calls by setting $Q(s, g, C_m) = -\infty$ for some appropriate set of (s, g, m) to reduce the search space. If such a restriction is too strong, performance will become worse. However, this does not make a task unsolvable because the execution of primitive actions is not restricted. An engineer may design appropriate restrictions of subroutine calls to tune overall performance, similar to the task graph design in the MAXQ architecture[6].

2.6 Action selection

The action-selection policy $\pi(\tilde{s}, \tilde{a})$ is derived from the action-value function $Q_G(\tilde{s}, \tilde{a})$. The current implementation uses a softmax action selection policy, which is defined as follows:

$$\pi((s, g), \tilde{a}) = \frac{exp(\beta Q_G((s, g), \tilde{a}))}{\Sigma_{\tilde{a}'} exp(\beta Q_G((s, g), \tilde{a}'))} = \frac{exp(\beta Q(s, g, \tilde{a}) + \beta V_G(g))}{\Sigma_{\tilde{a}'} exp(\beta Q(s, g, \tilde{a}') + \beta V_G(g))}$$
$$= \frac{exp(\beta Q(s, g, \tilde{a}))}{\Sigma_{\tilde{a}'} exp(\beta Q(s, g, \tilde{a}'))}. \tag{10}$$

2.7 Thought-mode

When learning complicated tasks, restricting the search space to include only combinations of simple subroutines makes learning faster. In this case, subroutines realize the temporal abstraction[5] of action sequences. In the RGoal architecture, a novel mechanism called thought-mode facilitates this behavior.

Suppose that the optimal routes between all neighboring pairs of landmarks have been already learned. Then, an approximate solution for the optimal route between distant landmarks can be obtained by connecting neighboring landmarks. For example, in Fig.1, the route $S \rightarrow m \rightarrow G_1$ is an approximate solution for the route from S to G_1. Such solutions can be found without taking any actions within the environment [2][3][4]. The thought-mode of RGoal is a mechanism for finding approximate routes by repeating simulations of episodes within an agent's brain. This mechanism can be implemented with only a few small modifications to the RGoal algorithm. Because of its simplicity, we consider this mechanism to be a promising first step toward a computational model for the planning mechanism of the human brain.

The behavior of thought-mode is described below. If the selected action $\tilde{a}$ is a primitive action, the simulated state in the agent's brain is *immediately* changed

```
1: procedure EPISODE(S, G, think-flag)
2:     s ← S;  g ← G;  stack ← empty
3:     Choose ã from s, g using policy derived from Q
4:     while s ≠ G do
5:         # Take action.
6:         if ã = RET then
7:             s′ ← s;  g′ ← stack.pop();  r ← 0
8:         else if ã is C_m then
9:             stack.push(g);  s′ ← s;  g′ ← m;  r ← R^C
10:        else
11:            if think-flag then
12:                s′ ← g;  g′ ← g;  r ← dummy
13:            else
14:                Take action ã, observe r, s′;  g′ ← g
15:        # Choose action.
16:        if s′ = g′ then
17:            ã′ ← RET
18:        else
19:            Choose ã′ from s′, g′ using policy derived from Q
20:        # Update.
21:        if s = g or (think-flag and ã is not C_m) then
22:            # Do nothing.
23:        else
24:            Q(s, g, ã) ← Q(s, g, ã) + α(r + Q(s′, g′, ã′) − Q(s, g, ã) + V_g(g′))
25:        s ← s′;  g ← g′;  ã ← ã′
```

Fig. 3. Pseudo code for the RGoal algorithm, which is based on the Sarsa algorithm.

from s to the current subgoal g. In such cases, the table element $Q(s, g, \tilde{a})$ is not updated. If $\tilde{a}$ is a subroutine call C_m, the behavior of thought-mode is the same as that of the normal mode. In such cases, the subgoal g is changed to m and the table element $Q(s, g, C_m)$ is updated normally.

The behavior described above can be regarded as a type of model-based reinforcement learning[1]. The learned value of $Q(s, g, \tilde{a})$ is used as a model of the environment that tells the agent how much reward will be obtained if the agent moves from s to g.

2.8 RGoal algorithm

The pseudo code for the RGoal algorithm, which is based on the Sarsa algorithm, is presented in Fig.3. This algorithm uses a flat table and stack with a single loop consisting entirely of simple operations.

3 Evaluation

RGoal performance was evaluated on a maze task. Here, we focus on convergence speed to suboptimal solutions, rather than exact solutions.

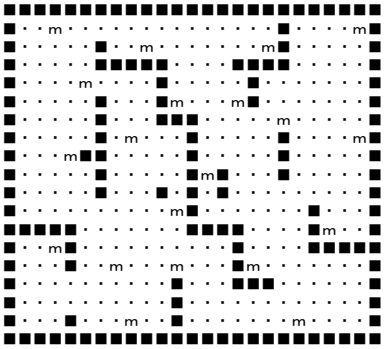

Fig. 4. Map of a maze on the 2D grid used for the evaluation. Twenty landmarks (denoted **m**) are placed on the map. For each episode, the start S and goal G are randomly selected from the landmark set.

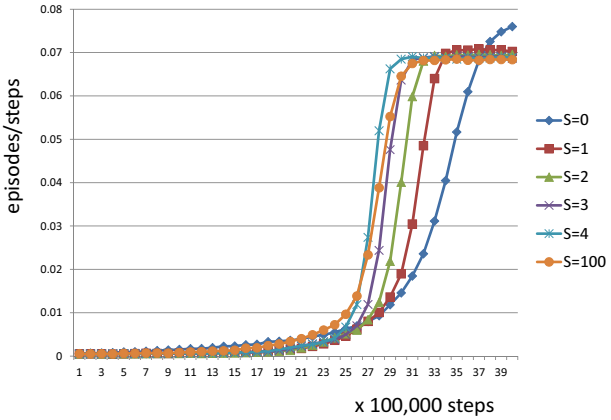

Fig. 5. Experiment 1. Relationship between the upper limit S of the stack depth and RGoal performance. $S = 0$ corresponds to non-hierarchical reinforcement learning. When $S = 1$, recursive calls are not allowed, as in the two-layered reinforcement learning. A greater upper limit results in faster convergence because it increases the opportunity for the reuse of subroutines.

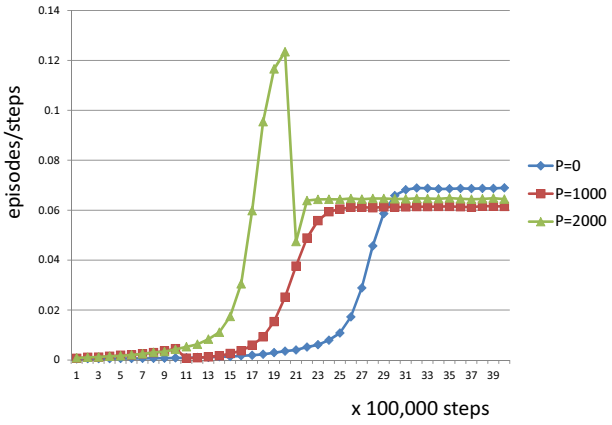

Fig. 6. Experiment 2. Relationship between the length P (times 1000 steps) of the pre-training phase and RGoal performance. In the pre-training phase, only pairs of the start and goal within Euclidean distances of eight are selected. $S = 100$. The graph also includes a change in the score during the pre-training phase. The greater the value of P, the faster the convergence speed.

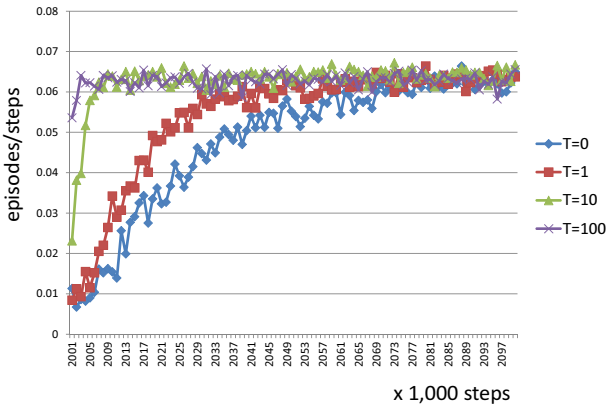

Fig. 7. Experiment 3. Relationship between thought-mode length T and RGoal performance. T is the number of simulations executed prior to the actual execution of each episode. $S = 100$, $P = 2000$. Here, we only plot the change in score after the pre-training phase. If thought-mode length is sufficiently long, approximate solutions are obtained in almost zero-shot time.

The map and landmark set are presented in Fig.4. For each episode, the start S and goal G are randomly selected from the landmark set. When the agent reaches G, the episode ends and the next episode with a different start and goal begins.

The reward for moving up, down, left, or right is -1, that for moving diagonally is $-\sqrt{2}$, that for hitting a wall is -1, and that for a subroutine call is $R^{\mathcal{C}} = -1$. As mentioned earlier, rewards are not discounted.

The table elements of $Q(s, g, \tilde{a})$ are initialized to zero if $s = g$ and $-50-n$ (n is small noise), otherwise. To make learning faster, subroutine calls are restricted to be executable only on landmarks by initializing some appropriate elements to $-\infty$, as described in Sec.2.5.

The action-selection function is a softmax function with $\beta = 1$. The learning rate is $\alpha = 0.1$.

For each of the following experiments, the average values of 10 trials were calculated. For each graph, the horizontal axis is the number of steps and the vertical axis is the number of episodes per step. The larger the value of the y-axis, the faster the agent reaches the goal. Here, "the number of steps" means the number of moves within the map or collisions with a wall. Subroutine calls, returns from subroutines, and execution steps in thought-mode are not included because they are regarded as virtual actions in the agent's brain.

Experiment 1 examined the relationship between the upper limit S of the stack depth and RGoal performance (Fig.5). When the stack depth reaches the upper limit, the agent does not make any further subroutine calls. $S = 0$ corresponds to non-hierarchical reinforcement learning. A greater upper limit results in faster convergence. However, at $S = 100$, the convergence is slightly slower than that at $S = 4$. The score after convergence is the best when $S = 0$. This is because if subroutines can be used, an agent may choose suboptimal routes through some landmarks. We have confirmed that even when subroutine calls can be used, if we optimize the search tendency by choosing a small value of β, then increasing β, the agent eventually finds the optimal policy that does not call subroutines.

Experiment 2 examined the relationship between the length P (times 1000 steps) of the pre-training phase and RGoal performance (Fig.6). In the pre-training phase, only pairs of the start S and goal G within Euclidean distances of eight are selected. Such pairs constitute 60 pairs out of the $20 \times 19 = 380$ total pairs. In this experiment, $S = 100$. For fair comparison, the graph includes changes in the score during the pre-training phase. The results show that a greater value of P results in faster convergence during the normal phase after the pre-training phase. This means that if an agent learns simple tasks first, learning difficult tasks becomes faster because the learned simple tasks can be reused as subroutines.

Experiment 3 examined relationship between thought-mode length T and RGoal performance (Fig.7). T is the number of simulations in an agent's brain that are executed immediately before each actual execution of an episode in the environment. In this experiment, $S = 100$ and $P = 2000$. Here, we only

plot changes in the score after 2,000,000 steps of the pre-training phase. The results show that if the thought-mode length is sufficiently long, approximate solutions for unknown tasks are obtained immediately (almost in zero-shot time) by combining knowledge from previously experienced simple tasks.

4 Related work

Unlike previous hierarchical reinforcement learning architectures, RGoal is unique in that the caller and callee relation between subroutines is not predefined, but is learned within the framework of reinforcement learning. We have integrated several important ideas that were proposed in previous papers into a single simple architecture to realize the desired RGoal features.

RGoal has a very similar structure to the Hierarchical Distance to Goal (HDG) architecture[4]. HDG uses a dedicated algorithm for offline searching of routes by connecting distant landmarks. In contrast, RGoal accomplishes the same goal by using thought-mode, which is much easier to implement and is similar to human behavior.

The H-DYNA architecture[2][3] also utilizes planning with temporal abstraction, similar to the thought-mode in our architecture.

MAXQ[6] is an architecture for hierarchical reinforcement learning that can utilize layers deeper than two and handles subtask sharing through value function decomposition. In RGoal, decomposition becomes simpler based on the assumption that each subroutine terminates in a single state.

The R-MAXQ architecture[8] introduced the feature of model-based reinforcement learning into MAXQ. It straightforwardly leans and utilizes a model of the environment. In RGoal, the learned $Q(s, g, \tilde{a})$ is used as a model of the environment.

Derivation of a hierarchical policy using an augmented state-action space was proposed in [9]. The space in RGoal is simpler and visually understandable, thereby facilitating easier understanding of recursive subgoal settings.

The option-critic architecture[11] acquires options (subroutines) from agent experiences. In RGoal, the landmark set is given or supposed to be acquired as salient states experienced by the agent.

Because the theoretical framework of RGoal is simple, it is easy to extend. For example, techniques for accelerating learning, such as universal value function approximators[10] or hindsight experience replay[12], should be easily applicable.

5 Conclusion

We proposed a novel hierarchical reinforcement learning architecture that allows unlimited recursive subroutine calls. We integrated several important ideas that were proposed in previous papers into a single simple architecture. A novel mechanism called thought-mode combines learned simple tasks to solve unknown complicated tasks rapidly, sometimes in zero-shot time. Because of its simplicity,

we consider RGoal to be a promising first step toward a computational model of the planning mechanism of the human brain. In the future, RGoal will be applicable to robots that purposefully use tools such as ladders. A dialogue system that makes purposeful speech is also one of the applications aimed at.

In the future, we will attempt to speed up learning by introducing state abstraction via function approximation and aim for more realistic application tasks. Detailed comparisons with other approaches are also important future work.

Acknowledgments We gratefully acknowledge Yu Kohno and Tatsuji Takahashi for their helpful discussion.

This work was supported by JSPS KAKENHI Grant Number JP18K11488.

References

1. Sutton, R. S., Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In Proceedings of the Seventh International Conference on Machine Learning,pp. 216–224 1990.
2. Singh, S. P., Reinforcement learning with a hierarchy of abstract models. In Proceedings of the Tenth National Conference on Artificial Intelligence, San Jose, California. AAAI Press. 202–207, 1992.
3. Singh, Satinder Pal, Scaling reinforcement learning algorithms by learning variable temporal resolution models. In Proceedings of the Ninth International Conference on Machine Learning, Aberdeen, Scotland. Morgan Kaufmann. 406–415, 1992.
4. Kaelbling, L. P., Hierarchical Learning in Stochastic Domains: Preliminary Results. In Proceedings of the 10th International Conference on Machine Learning, San Francisco, California. Morgan Kaufmann. 167–173, 1993.
5. Sutton, R. S., Precup, D., and Singh, S. P., Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence 112(1-2), 181–211, 1999.
6. Thomas G. Dietterich, Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition, Journal of Artificial Intelligence Research 13, 227–303, 2000.
7. Wiewiora, E., Potential-based shaping and Q-value initialization are equivalent, Journal of Artificial Intelligence Research 19, 205-208, 2003.
8. Jong, N. and Stone, P., Hierarchical model-based reinforcement learning: R-Max + MAXQ. In Proceedings of ICML, 2008.
9. Levy, K. Y., and Shimkin, N., Unified inter and intra options learning using policy gradient methods. In Proceedings of The 9th European Workshop on Reinforcement Learning (EWRL-9), 153–164, 2011.
10. Schaul, T., Horgan, D., Gregor, K., and Silver, D., Universal value function approximators. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15), 1312–1320, 2015.
11. Bacon, P.-L., Harb, J., and Precup, D., The option-critic architecture. In Proceedings of AAAI, 1726–1734, 2017.
12. Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R., Welinder, P. , McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W., Hindsight experience replay, Advances in Neural Information Processing Systems 30, pp. 5055–5065, 2017.