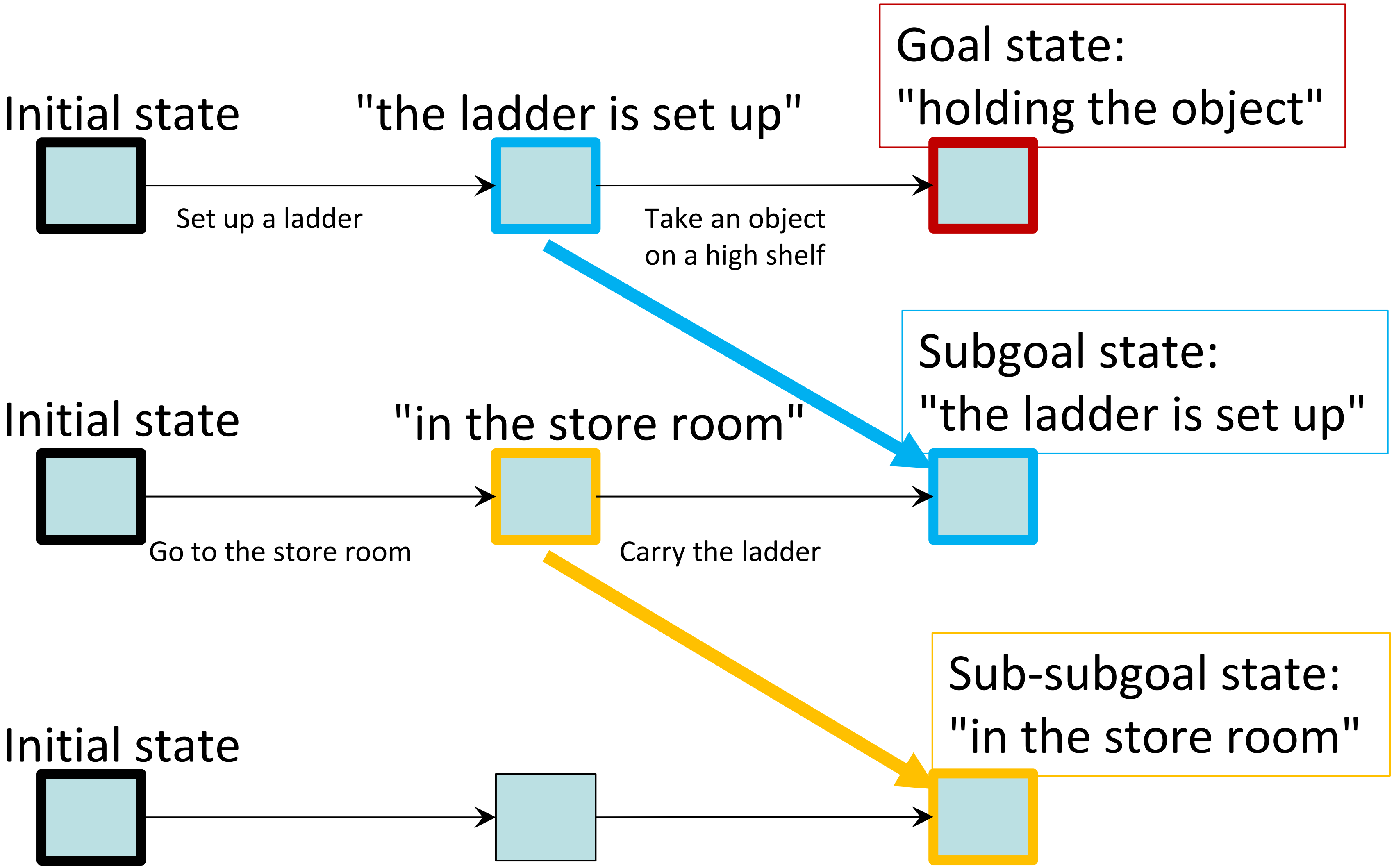
Hierarchical Reinforcement Learning with Unlimited Recursive Subroutine Calls

2019-09-17 ICANN 2019

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

Humans can set suitable subgoals recursively to achieve certain tasks



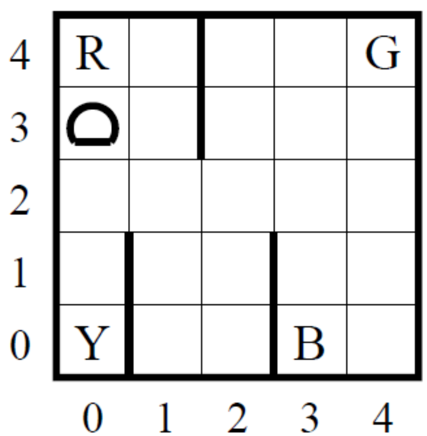
The depth of this recursion is apparently unlimited.

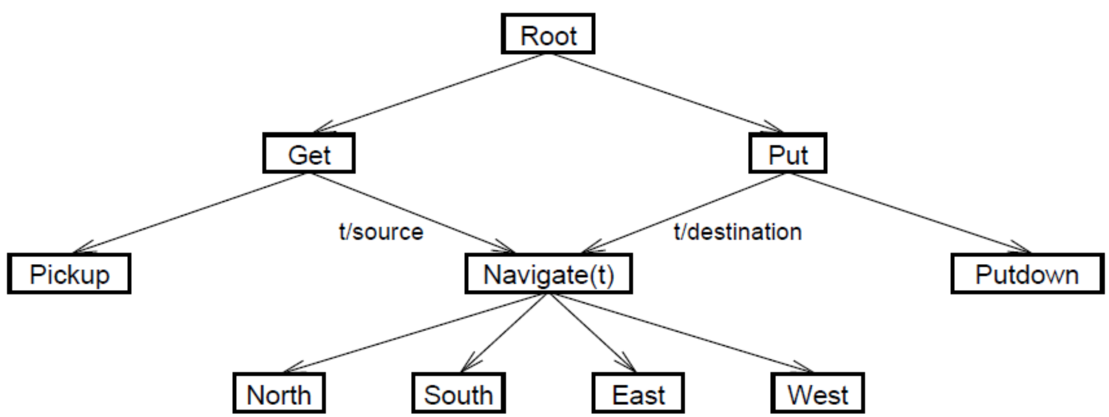
Hierarchical reinforcement learning architecture RGoal

- 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 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.

Previous work: MAXQ [Dietterich 2000]

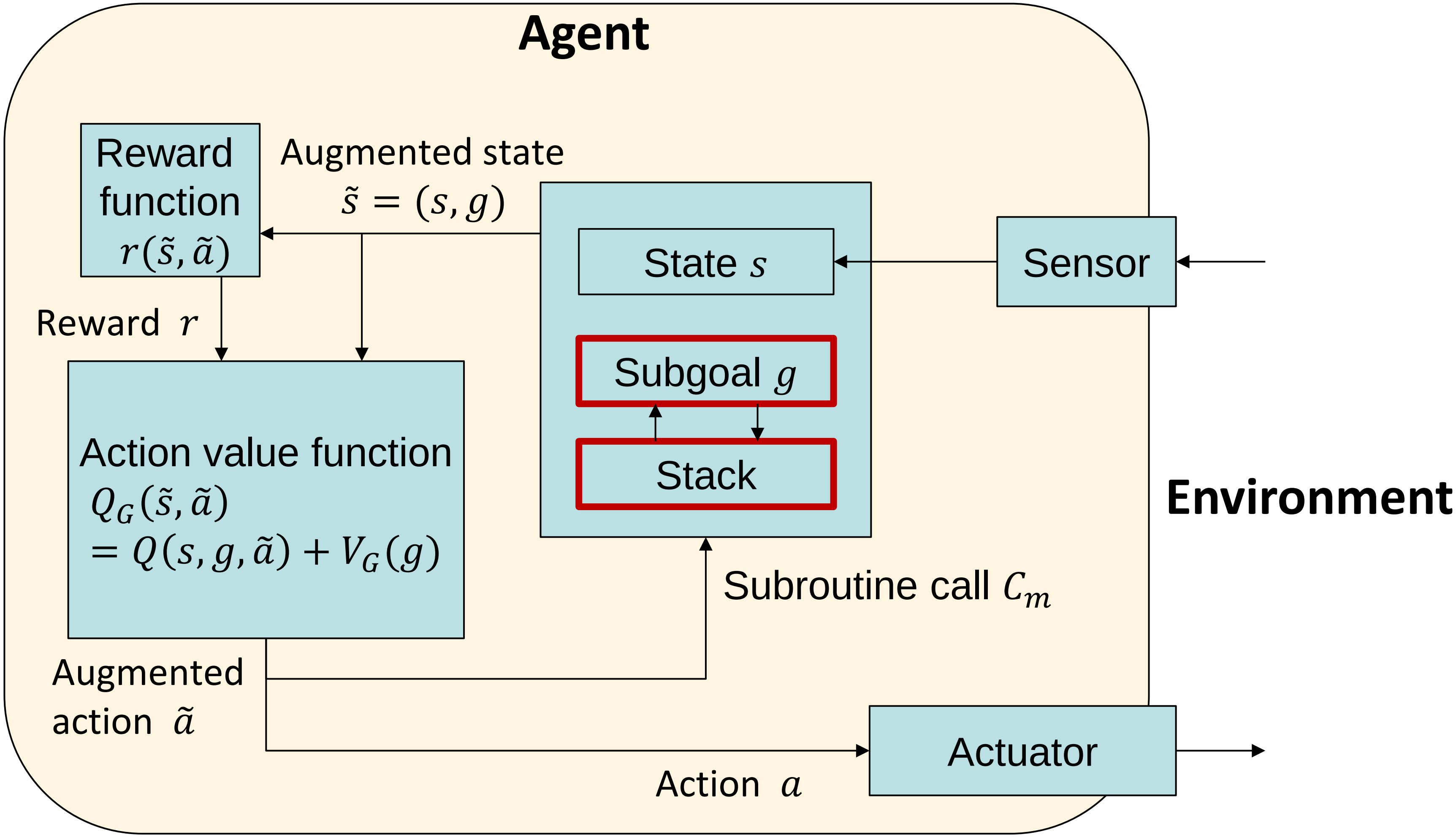
- Multi-layered hierarchical reinforcement learning architecture with a **fixed number** of layers.
- Accelerates learning speed
  - Subtask sharing, Temporal abstraction, State abstraction





RGoal simplifies MAXQ archicecture and extends its feature.

RGoal architecture



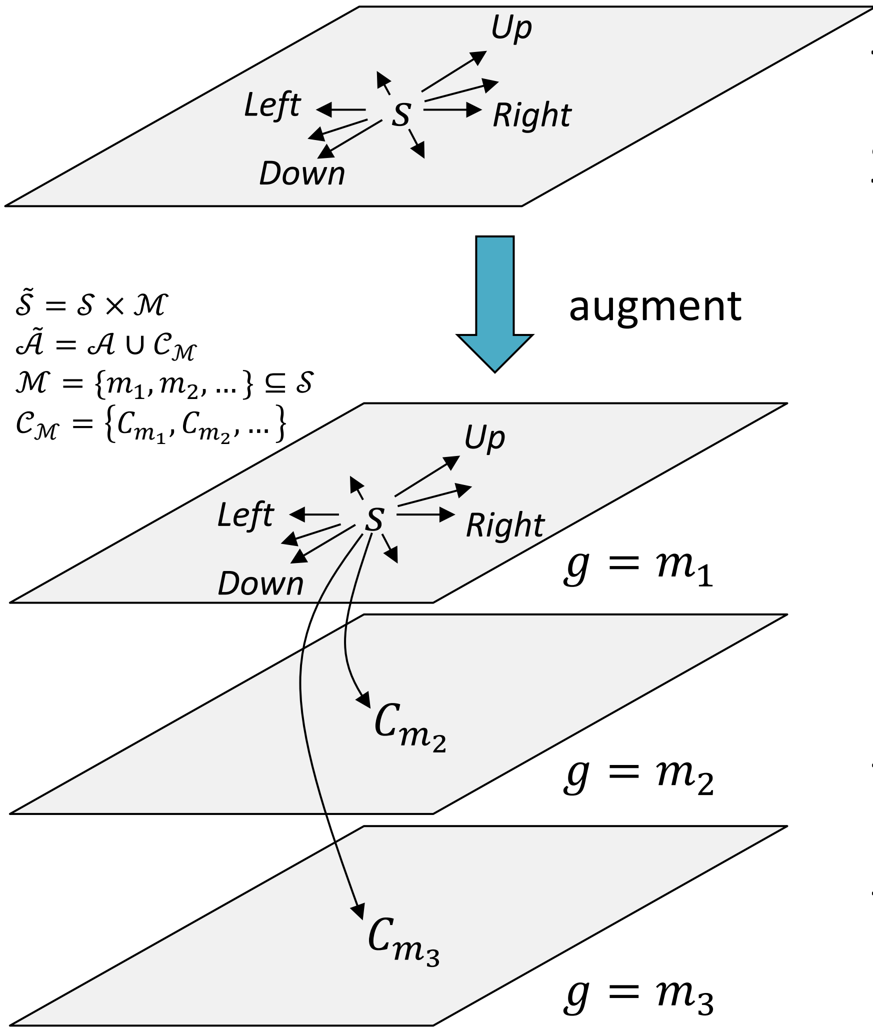
The agent has a subgoal and a stack as its internal states.

Augmented state-action space

[Levy and Shimkin 2011]

$\mathcal{S} = \{(0,0), (0,1), \dots\}$
$\mathcal{A} = \{Up, Down, Right, Left, \dots\}$

$\tilde{\mathcal{S}} = \mathcal{S} \times \mathcal{M}$
$\tilde{\mathcal{A}} = \mathcal{A} \cup \mathcal{C}_{\mathcal{M}}$
$\mathcal{M} = \{m_1, m_2, \dots\} \subseteq \mathcal{S}$
$\mathcal{C}_{\mathcal{M}} = \{C_{m_1}, C_{m_2}, \dots\}$



The augmented state is a pair of state and goal $\tilde{s} = (s, g)$.
The augmented action $\tilde{a}$ is a primitive action a or a subroutine call C_m .

RGoal solves the Markov Decision Process (MDP) in the augmented state-action space.

Unlike previous hierarchical RLs, caller and callee relation between subroutines is not predefined, but is learned within the framework of RL.
(Subroutine call is just a state transition in the augmented state-action space.)

Value function decomposition

[Singh 1992][Dietterich 2000]

- Action value function Q is decomposed into two parts:

$$Q_G^{\pi}((s,g),a) = \underbrace{Q^{\pi}(s,g,a)}_{\text{before subgoal}} + \underbrace{V_G^{\pi}(g)}_{\text{after subgoal}}$$

where $V_G^{\pi}(g) = \Sigma_a \pi\big((g,G),a\big) Q^{\pi}(g,G,a)$
G : original global goal
g : current subgoal

- $Q(s,g,a)$ can be **shared between tasks** because it does not depend on the original goal G.
  - Accelerates learning speed

Update rule of Q(s,g,a) is derived from the standard Sarsa algorithm

$$Q(\tilde{s},\tilde{a}) \leftarrow Q(\tilde{s},\tilde{a}) + \alpha(r + Q(\tilde{s}',\tilde{a}') - Q(\tilde{s},\tilde{a}))$$

When subgoal is g and stack contents are g1,g2,...gn,
the agent moves though the path: s -> g -> g1 -> g2 -> ... -> gn .
If a subroutine g' is called, the changed path is: s -> g' -> g -> g1 -> g2 -> ... -> gn .
Therefore, the equation bellow holds:

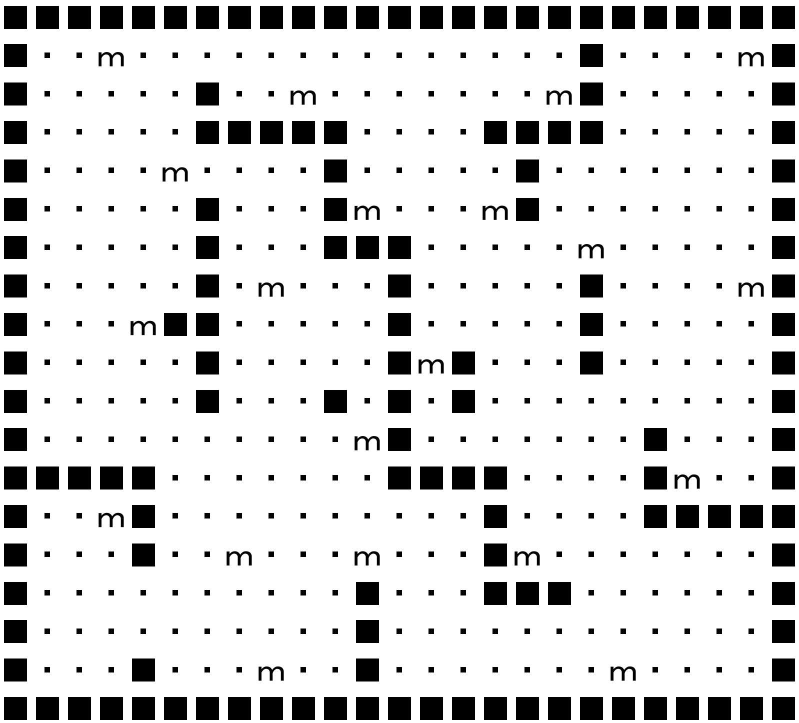
$$\begin{aligned}
&Q_{g_n}(\tilde{s}',\tilde{a}') - Q_{g_n}(\tilde{s},\tilde{a}) \\
&= \left(Q(s',g',\tilde{a}') + V_g(g') + V_{g_1}(g) + V_{g_2}(g_1) + \cdots + V_{g_n}(g_{n-1})\right) \\
&\quad - \left(Q(s,g,\tilde{a}) + V_{g_1}(g) + V_{g_2}(g_1) + \cdots + V_{g_n}(g_{n-1})\right) \\
&= Q(s',g',\tilde{a}') - Q(s,g,\tilde{a}) + V_g(g')
\end{aligned}$$

This equation also holds when $\tilde{a}$ is not a subroutine call,
but is a primitive action. Therefore:

$$Q(s,g,a) \leftarrow Q(s,g,a) + \alpha(r + Q(s',g',a') - Q(s,g,a) + V_g(g'))$$

Efficiently executed.

Maze task



m : landmark

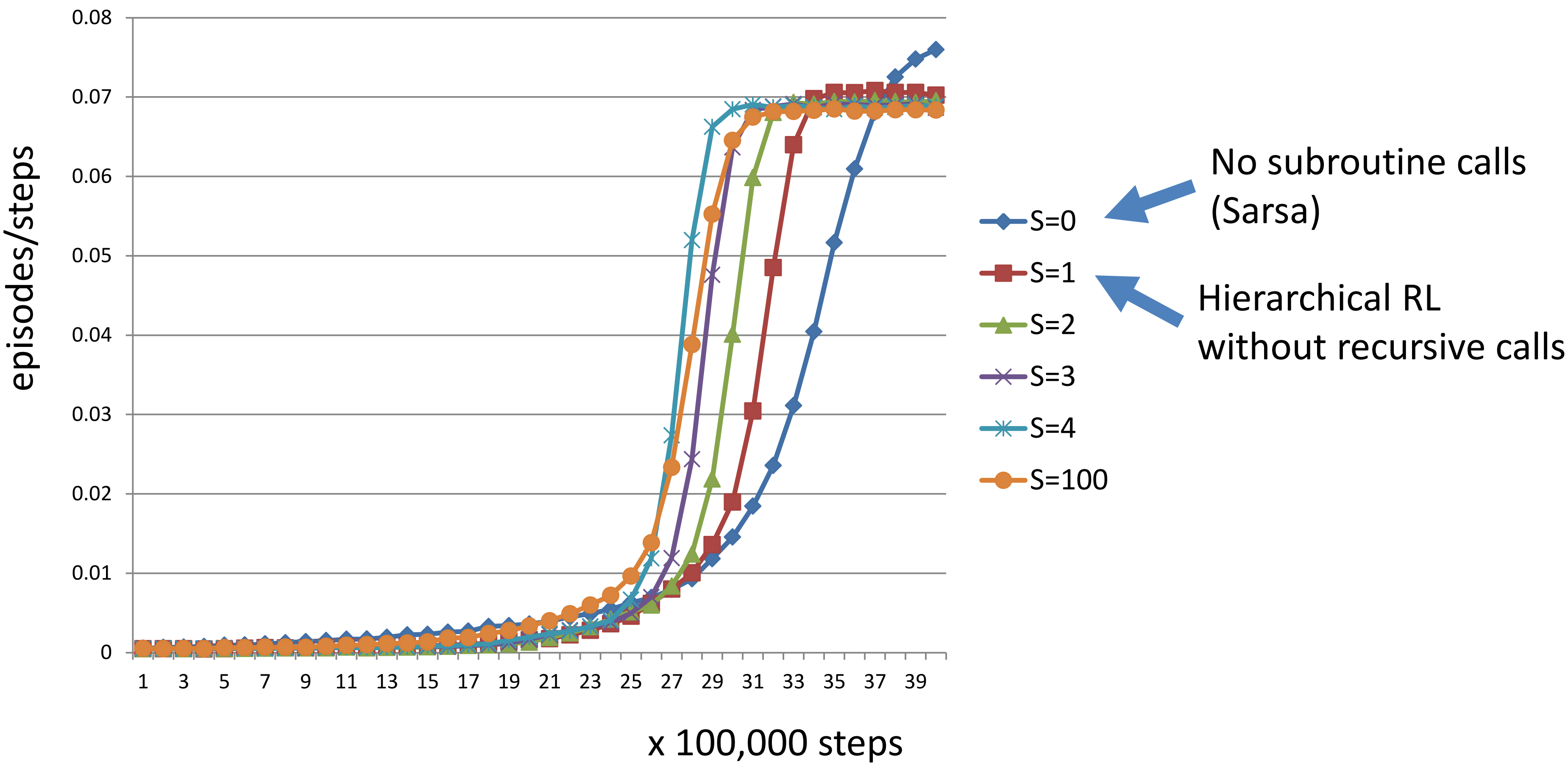
Twenty **landmarks** are placed on the map.

Landmarks may become subgoals.

For each episode, the start S and goal G are randomly selected from the landmark set.

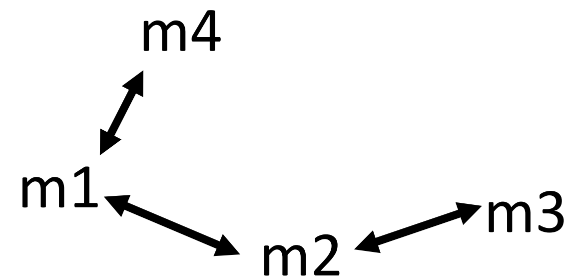
We focus on convergence speed to suboptimal solutions, rather than exact solutions.

Experiment 1. Relationship between the upper limit S of the stack depth and RGoal performance

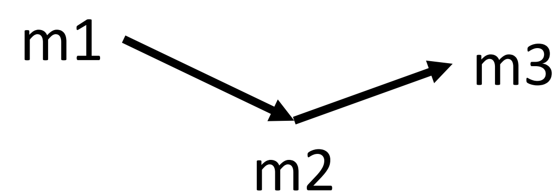


A greater upper limit results in **faster convergence**
because it increases the opportunity for the reuse of subroutines.

"Thought-mode" combines learned simple tasks to solve unknown complicated tasks

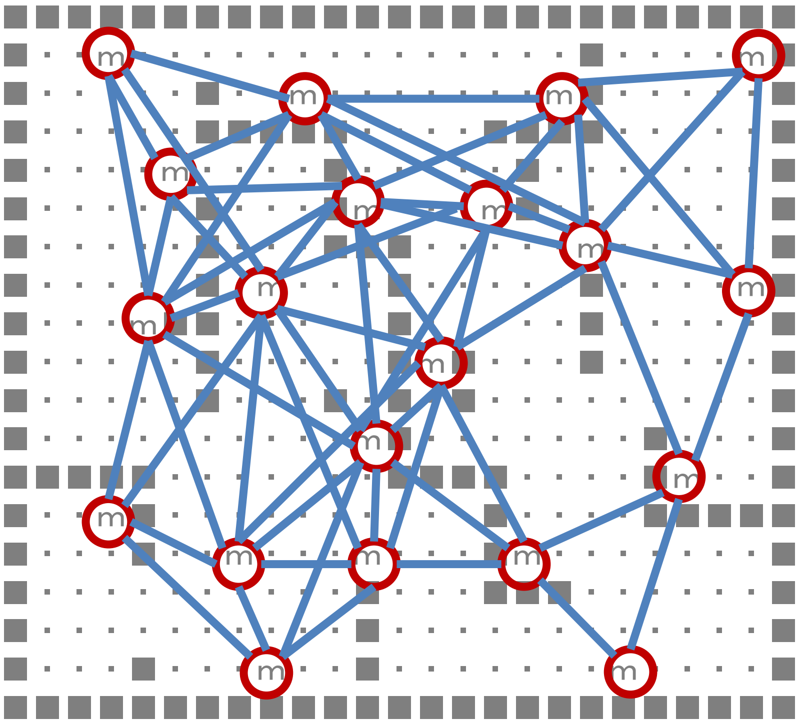


- Suppose that the optimal routes between all neighboring pairs of landmarks have been already learned.
- An approximate solution for the optimal route between distant landmarks can be obtained by connecting neighboring landmarks.



- **Such solutions can be found without taking any actions within the environment.** [Singh 1992]
  – Found by simulations within the agent's brain
    - A kind of Model-based RL
    - A kind of Deductive reasoning

Before evaluation of Thought-mode, the agent learns simple tasks

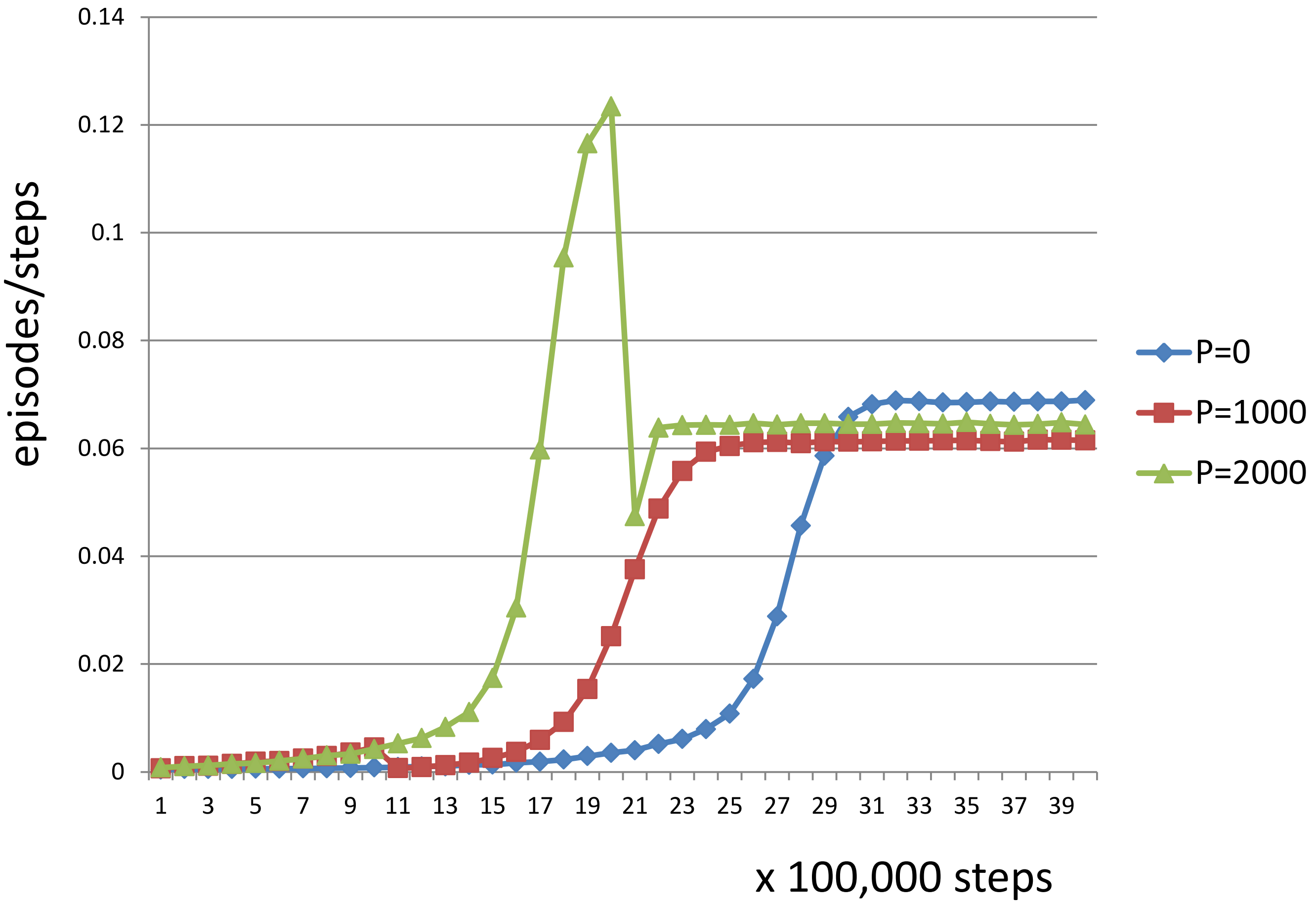


m : landmark

In the pre-training phase, only pairs of the start and goal within Euclidean distances of eight are selected.

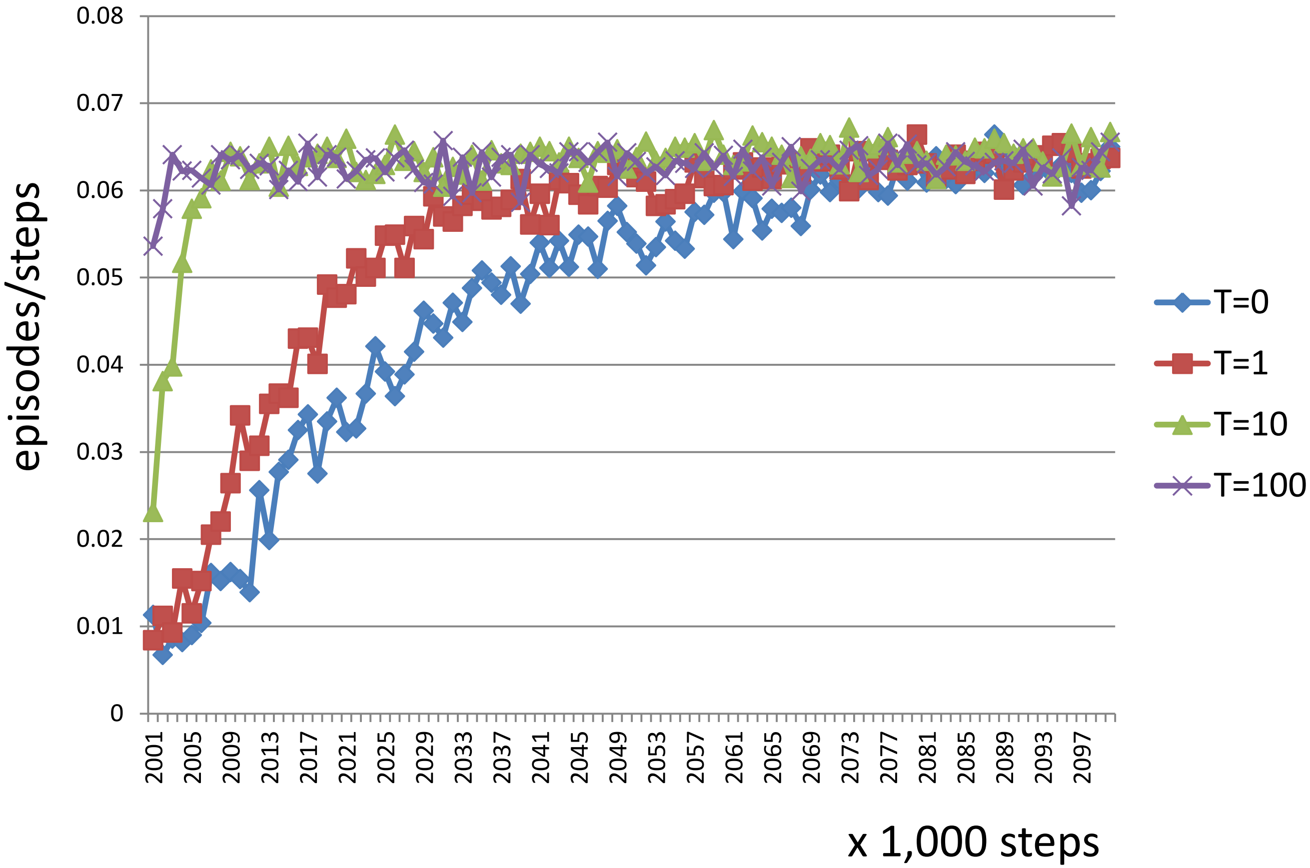
In the evaluation phase, arbitrary pairs are selected.

Experiment 2. Relationship between the length P of the pre-training phase and RGoal performance



A greater value of P results in faster convergence
If an agent learns simple tasks first, learning difficult tasks becomes faster because the learned simple tasks can be reused as subroutines.

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.

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**.

Pseudo code of RGoal

```
procedure EPISODE(S, G, think-flag)
    s ← S;  g ← G;  stack ← empty
    Choose ã from s, g using policy derived from Q
    while s ≠ G do
        # Take action.
        if ã = RET then
            s′ ← s;  g′ ← stack.pop();  r ← 0
        else if ã is C_m then
            stack.push(g);  s′ ← s;  g′ ← m;  r ← R^C
        else
            if think-flag then
                s′ ← g;  g′ ← g;  r ← dummy
            else
                Take action ã, observe r, s′;  g′ ← g
        # Choose action.
        if s′ = g′ then
            ã′ ← RET
        else
            Choose ã′ from s′, g′ using policy derived from Q
        # Update.
        if s = g or (think-flag and ã is not C_m) then
            # Do nothing.
        else
            Q(s, g, ã) ← Q(s, g, ã) + α(r + Q(s′, g′, ã′) − Q(s, g, ã) + V_g(g′))
        s ← s′;  g ← g′;  ã ← ã′
```

This algorithm only uses **simple data structures** and a **simple single loop**.

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.

Conclusion

- We proposed a novel hierarchical RL architecture that allows **unlimited recursive subroutine calls**.
- We integrated several important ideas proposed before into **a single simple architecture**.
  - Augmented action-value space[Levy and Shimkin 2011],
  - Value function decomposition[Singh 1992][Dietterich 2000]
  - Model-based RL [Sutton 1990]
- A novel mechanism called **Thought-mode** combines learned simple tasks to solve unknown complicated tasks rapidly, **sometimes in zero-shot time**.
- Because the theoretical framework and the architecture of RGoal is simple, it is easy to extend.