社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

Fusing Learning Strategies to Learn Various Tasks with Single Configuration

Akihiko YAMAGUCHI†, Jun TAKAMATSU†, and Tsukasa OGASAWARA†

† Graduate School of Information Science, Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara 630-0192, JAPAN

Abstract This paper proposes a method to fuse learning strategies (LSs) in reinforcement learning framework. Generally, we need to choose a suitable LS for each task respectively. In contrast, the proposed method automates this selection by fusing LSs. The LSs fused in this paper includes a transfer learning, a hierarchical RL, and a model based RL. The proposed method has a wide applicability. When the method is applied to a motion learning task, such as a crawling task, the performance of motion may be improved compared to an agent with a single LS. The method also can be applied to a navigation task by hierarchically combining already learned motions, such as a crawling and a turning. This paper demonstrates a maze task of a humanoid robot where the robot learns not only a path to goal, but also a crawling and a turning motions.

Key words Learning System, Reinforcement Learning, Modularization, Motion Learning, Humanoid Robot

1. Introduction

Designing a behavior by only its objective is essential for future robots, since this ability enables the end-users to teach their wish to the robots easily. Reinforcement Learning (RL) method is such a technology. There are a lot of RL applications to robotics research [1]〜[4]. However, RL methods require a lot of learning cost in large domains, such as motion learning of a humanoid robot.

Many researchers have tackled this issue and proposed effective methods; for instance, dimension reduction [2], hierarchical RL [3], transfer learning [5], model utilization [6], imitation learning [4]. We refer to such a method as a *learning strategy (LS)*.

These LSs improve RL, however, their effectiveness or applicability depends on a task. For instance, a dimension reduction by using a pattern generator may improve cyclic motions; meanwhile, this way may restrict the capability to learn episodic motions such as jumping. The dependency of LSs on tasks means that the end-users should select a proper LS for each task. This selection may be difficult for ordinary users.

Thus, we aim to make a system in which *multiple* LSs are applied to each task of a single robot *multiple* times, where the ordering of the LSs is automatically decided. For this purpose, the highly modularized learning system mainly consists of the following four elements:

Behavior modules: deciding the behavior of a robot.

Fundamental module types: constructing behavior modules, such as a module of an RL method.

LS modules: generating behavior modules from the fundamental module types according to its learning strategy.

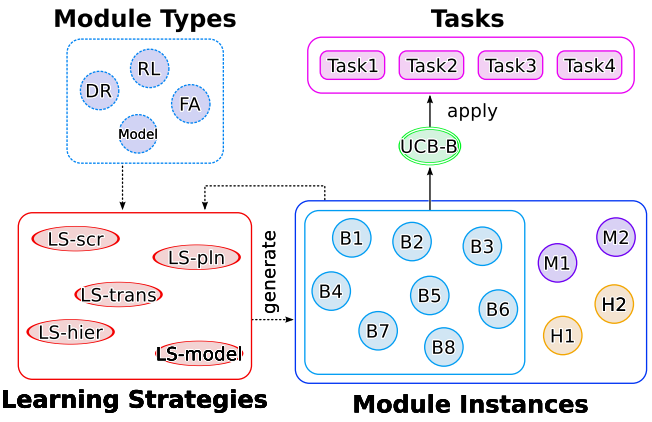

Fig. 1 Overview of the highly modularized learning system. Every small circles, ellipses, and rectangles indicate modules. UCB-B denotes UCB-Boltzmann selection method.

UCB-Boltzmann selection method: choosing a behavior module actually used in each learning stage.

Fig. 1 illustrates the overview of the system. The system maintains a set of module types. The LS modules generate behavior modules from the types (Module Instances). Each behavior module is used to decide a behavior of the robot and learn a task. Some of the LS modules generate behavior modules from existing behavior modules (e.g. a transfer LS). The system has multiple behavior modules for each task. Each behavior module has upper confidence bound (UCB) as an evaluation value which is calculated from reward observation. In each learning stage, a behavior module that actually decides the movement of the robot is chosen from the whole set of behavior modules by UCB-Boltzmann selection method. UCB-Boltzmann selection method is Boltzmann

selection method with UCB, which enables a probabilistic exploration. The selected behavior module is trained during the learning stage. Thus, each behavior module is generated through a sequence of LSs, and the selection of behavior modules means the selection of a proper LS sequence for the task. The core algorithm is referred to as *LS fusion* method.

This paper also defines the following LSs:

LS-scratch generates a behavior module that learns a task from scratch using WF-DCOB [1].

LS-accelerating generates a behavior module by accelerating the motion of a source behavior module.

LS-freeing generates a behavior module by increasing the DoF of a source behavior module.

LS-planning generates a behavior module that uses a model-based RL method such as Dyna [6].

LS-hierarchy generates a hierarchical action space module.

Here, the LS-accelerating and the LS-freeing are *transfer learning* methods [5]. A remarkable feature of these transfer LSs is that the LSs transfers not only policy parameters, but also physical limitation of the policy.

The proposed LS fusion is applied to a maze task of the simulated humanoid robot where the robot learns not only a path to goal, but also a crawling and a turning motions. The rest of this paper is organized as follows. Section 2. describes LS fusion method, Section 3. defines the LSs, Section 4. demonstrates the experiments, and Section 5. concludes this paper.

2. Learning Strategy Fusion

LS fusion consists of two elements: (1) the LSs, and (2) UCB-Boltzmann selection. There are two types in LSs; *behavior* LSs that generate new behavior modules, and *supplementary* LSs that generate supplementary modules other than behaviors, such as model modules and hierarchical-action-space modules. Let $\mathcal{LS}_{\rm bhv}$ denote the set of the behavior LSs, $\mathcal{LS}_{\rm spl}$ denote the set of the supplementary LSs. UCB-Boltzmann selection method chooses a behavior from both the existing behavior modules and the new ones generated by the behavior LSs. The selected behavior module is used to actually control the robot, and the module is updated its policy from samples.

We design LS fusion so that the behavior and the supplementary modules are generated and learned through the following flow:

(1) The supplementary LSs generate modules if applicable.

(2) The behavior LSs generate new behavior modules. Specifically, the LS-scratch generates new behaviors which may have different DoF configurations. If there are behaviors trained enough, the LS-freeing and the LS-accelerating generate new behavior modules by transferring the trained behaviors.

(3) UCB-Boltzmann selection method chooses a behavior module from both the existing behavior modules and the new ones generated in (2).

(4) Several episodes are performed using the selected behavior module, and the policy of the behavior module is updated from samples. The supplementary modules are also updated if possible.

(5) (1)...(4) are repeated.

Here, UCB (upper confidence bound) uses both the mean of a reward summation $\overline{R}_B$ and its deviation $\overline{\sigma}_B$. The deviation $\overline{\sigma}_B$ can estimate the potential improvement of the performance. The deviation $\overline{\sigma}_B$ is also used to judge if a behavior module is trained enough.

2.1 LS Fusion Algorithm

We assume that several pairs of a control command space and a state space $\{(\tilde{\mathcal{U}}, \mathcal{X})\}$ are predefined; they have different DoF configurations. Here, defining $\tilde{\mathcal{U}}$ and $\mathcal{X}$ means giving conversions between $(\tilde{\mathcal{U}}, \mathcal{X})$ and the overall (full DoF) command and state spaces $(\tilde{\mathcal{U}}_{\rm w}, \mathcal{X}_{\rm w})$. Specifically, we assume linear conversions with constant matrices $\mathrm{C}_{\tilde{\mathcal{U}}}$ and $\mathrm{C}_{\mathcal{X}}$ such that $\tilde{u}_{\rm w} = \mathrm{C}_{\tilde{\mathcal{U}}}\tilde{u}$, $x = \mathrm{C}_{\mathcal{X}} x_{\rm w}$ where $\tilde{u} \in \tilde{\mathcal{U}}$, $\tilde{u}_{\rm w} \in \tilde{\mathcal{U}}_{\rm w}$, $x \in \mathcal{X}$, and $x_{\rm w} \in \mathcal{X}_{\rm w}$.

Each behavior learning strategy LS is defined as a function $\mathsf{GEN}_{\rm bhv}(LS, \mathcal{U}, \mathcal{X}, \mathit{Task})$ that generates behavior modules, and each supplementary learning strategy is defined as a function $\mathsf{GEN}_{\rm spl}(LS, \mathit{Task})$ that generates supplementary modules.

The LS fusion algorithm is defined for an episodic task. Algorithm 1 shows the overall algorithm(*1). Here, $\mathrm{N}_{\rm LSSp}$ is an interval of executing the supplementary LSs (LSSp means LS Supplementary), $\mathrm{N}_{\rm LSBh}$ is an interval of executing the behavior LSs (LSBh means LS Behavior). $\mathrm{N}_{\rm LSBh} > 1$ is needed to compute the valid reward statistics (we choose $\mathrm{N}_{\rm LSSp} = 20$ and $\mathrm{N}_{\rm LSBh} = 10$ in the experiments). UCB-Boltzmann selection method chooses a behavior module from both the existing $\mathcal{B}$ and new behavior modules generated by the behavior LSs. Note that only the selected new behavior module is added into $\mathcal{B}$.

Thus, the key element of LS fusion is each LS ($\mathsf{GEN}_{\rm bhv}$, $\mathsf{GEN}_{\rm spl}$) and UCB-Boltzmann selection method. Note that LS fusion works with any LS for that $\mathsf{GEN}_{\rm bhv}$ or $\mathsf{GEN}_{\rm spl}$ is defined. The rest of this section describes the reward statistics and UCB-Boltzmann selection method. In the next section, we specify the LSs used in this thesis.

2.2 Reward Statistics

We evaluate the performance of a behavior module by $R \triangleq \frac{\sum_t r_t}{T}$, where $\{r_t | t = 1, 2, \dots\}$ denotes the observed reward sequence in an episode, and T denotes total time in the episode. The definition of R depends on a task. In general, a sum of reward (return) may be used, but in our crawling task, this definition is suitable to select a better behavior module, especially in the early stage of learning.

(*1) : In implementing this algorithm, the size of $\mathcal{B}$ is limited to 20 per a task to prevent the large memory usage. If the size exceeds the limit, a behavior module that has the minimum UCB of $R_{\rm UCB}$ except for $B_{\rm next}$ is removed from $\mathcal{B}$.

Algorithm 1: Learning strategy fusion

Input: Task *Task*, behavior modules $\mathcal{B}$,
state-space modules $\{\mathcal{X}\}$, action-space modules: $\{\mathcal{U}\}$,
dynamics-model modules $\{M_{\rm dyn}\}$, reward-model modules $\{M_{\rm rwd}\}$
/* $\{M_{\rm dyn}\}$ and $\{M_{\rm rwd}\}$ may be empty */

```
for N_eps = 1, 2, ... do /* N_eps: episode number */
  if N_eps mod N_LSSp = 0 then
    for each LS ∈ LS_spl do
      {X}, {U}, {M_dyn}, {M_rwd} ← GEN_spl(LS, Task)
  if N_eps mod N_LSBh = 0 then
    /* select a behavior module: */
    B_new ← {}
    for each (U, X) do
      for each LS ∈ LS_bhv do
        B_new ← B_new ∪ GEN_bhv(LS, U, X, Task)
    Select B_next from B ∪ B_new by UCB-Boltzmann selection
    if B_next ∈ B_new then B' ← B ∪ {B_next} else B' ← B
    return B_next, B'
  Perform the episode with B_next:
    B_next is updated by its own learning algorithm
    {M_dyn}, {M_rwd} are updated if possible
  Update the reward statistics R̄_{B_next}, R̄²_{B_next}, σ̄_{max B_next}
```

Since each behavior module is updated to obtain better policy, its performance changes with episodes. Thus, we compute the mean and the standard deviation of R with forgetting the old data, and use them to select a behavior module. Let $R_{N_{\rm eps}}$ the observation at an $N_{\rm eps}$-th episode. The reward statistics $\overline{R}_B$, $\overline{R^2}_B$ are updated by

$$\overline{R}_B \leftarrow \alpha_{\rm R} R_{N_{\rm eps}} + (1-\alpha_{\rm R})\overline{R}_B, \tag{1}$$

$$\overline{R^2}_B \leftarrow \alpha_{\rm R} R^2_{N_{\rm eps}} + (1-\alpha_{\rm R})\overline{R^2}_B, \tag{2}$$

where $\alpha_{\rm R}$ is a learning rate. The standard deviation of R can be obtained by $\overline{\sigma}_B = (\overline{R^2}_B - \overline{R}^2_B)^{1/2}$. Additionally, at the end of each episode, $\overline{\sigma}_{{\rm max}B}$ is updated which is used in some LSs.

The reward statistics $\overline{R}_B$, $\overline{R^2}_B$ are initialized by zero if B is generated by the LS-scratch. If B is generated by the LS-accelerating or the LS-freeing, the statistics are initialized by the source behavior module's values.

2.3 UCB-Boltzmann Selection

We employ UCB of R to evaluate the priority of search. Additionally, we use Boltzmann selection to probabilistically select a behavior module.

The UCB of R is defined by

$$R_{{\rm UCB}B} \triangleq \overline{R}_B + {\rm f}_{\rm UCB}\overline{\sigma}_B \tag{3}$$

where ${\rm f}_{\rm UCB}$ is a real constant value that decides the weight of expected improvement (typically 1 or 2).

According to Boltzmann selection, the probability to select B is defined as

$$\pi(B) \propto \exp\left(\frac{1}{\tau_{\rm lsd}} R_{{\rm UCB}B}\right) \tag{4}$$

where $\tau_{\rm lsd}$ is a temperature parameter to adjust randomness. We decrease $\tau_{\rm lsd}$ with $\tau_{\rm lsd} = \tau_{\rm lsd0}\exp(-\delta_{\tau_{\rm lsd}} N_{\rm eps})$.

3. Learning Strategies

This section defines the LSs, namely, defines a function $\mathsf{GEN}_{\rm bhv}(LS, \mathcal{U}, \mathcal{X}, Task)$ or $\mathsf{GEN}_{\rm spl}(LS, Task)$ for each learning strategy LS. Every behavior module B has information, $Task^{(B)}$: a task that B is learning, $LS^{(B)}$: a learning strategy with which the behavior module is generated, $\mathcal{U}^{(B)}$: an action space and $\mathcal{X}^{(B)}$: a state space where the behavior module learns, and $\mathcal{K}^{(B)}$: a set of BFs.

The LS-scratch requires that an RL method is (1) efficient in a learning-from-scratch case, and (2) suitable to define the LS-accelerating and the LS-freeing. Thus, we utilize WF-DCOB [1] for these LSs, since WF-DCOB is efficient in a learning-from-scratch case like a discrete action set but explores continuous actions stably. Additionally, WF-DCOB explicitly has parameters related to a speed of motion and target joint angles, which is suitable for the LS-accelerating and the LS-freeing.

In WF-DCOB, a continuous action space is generated from a set of BFs given to approximate a value function, then wire-fitting [7] is utilized to learn over the continuous action space. An element in the continuous action space is denoted as $u = (g, q^{\rm trg})$, where $g \in \mathbb{R}$ is called an *interval factor* that decides a speed of motion, and $q^{\rm trg} \in \mathcal{Q}$ is the target joint angles of a trajectory. WF-DCOB's configurations are $C_{\rm p}(x)$: a function that extracts $q \in \mathcal{Q}$ from a state $x \in \mathcal{X}$, $C_{\rm d}(x)$: a function that extracts the derivative of $q \in \mathcal{Q}$ from a state $x \in \mathcal{X}$ as $\dot{q} = C_{\rm d}(x)$, and $\mathcal{I}_{\mathcal{R}}$: a set of ranges of the interval factors. The action space defined by these configurations is learned using wire-fitting. Wire-fitting is an interpolator whose parameters are $\Theta^\top = (\theta_1^\top, U_1^\top, \theta_2^\top, U_2^\top, \ldots, \theta_{|\mathcal{W}|}^\top, U_{|\mathcal{W}|}^\top)$. Here, θ_i encodes an action value, and U_i encodes an action. The LS-accelerating and the LS-freeing are realized by modifying the action parameters $\{U_i\}$. Please see [1] for the detail of WF-DCOB.

3.1 LS-Scratch

The function $\mathsf{GEN}_{\rm bhv}(\mathsf{LS\text{-}scr}, \mathcal{U}, \mathcal{X}, Task)$ generates a behavior module if $\mathcal{U}$ is a command space $\tilde{\mathcal{U}}$ and there is no behavior module of the same setup. Namely, a module is generated if $\mathcal{B}$ does not include a behavior module B such that $LS^{(B)} = \mathsf{LS\text{-}scr}$, $\mathcal{U}^{(B)} = \tilde{\mathcal{U}}$, and $\mathcal{X}^{(B)} = \mathcal{X}$. The reason of preventing a generation of the same setup is that the probability that the new behavior module obtains better performance than the existing one is not high. A new behavior module $B_{\rm new}$ uses WF-DCOB with Q(λ)-learning, where a default (predefined) set of BFs is employed. If a default set of BFs is not predefined for $\mathcal{X}$, a new behavior module is not generated(*2). WF-DCOB's configurations $C_{\rm p}$, $C_{\rm d}$, *Ctrl*, $\mathcal{I}_{\mathcal{R}}$ are assumed to be predefined for each $(\tilde{\mathcal{U}}, \mathcal{X})$. Since this behavior module learns from scratch, the parameters of wire-fitting are initialized as the WF-DCOB's manner.

(*2) : In the following experiments, $\mathcal{X}_{16}$ is the case.

3.2 LS-Accelerating

The LS-accelerating generates behavior modules from source behavior modules that have the same command and state spaces. Also, the generation with the same setup is prevented, and this LS works with only a command space. The acceleration is performed by multiplying the interval factor of the source module's WF-DCOB by a real constant value $\mathrm{f}_{\mathrm{accel}} < 1$.

Specifically, in the function $\mathsf{GEN}_{\mathrm{bhv}}(\mathsf{LS\text{-}accel}, \tilde{\mathcal{U}}, \mathcal{X}, \mathit{Task})$, for each B_{src} such that $\mathcal{U}^{(B_{\mathrm{src}})} = \tilde{\mathcal{U}}$ and $\mathcal{X}^{(B_{\mathrm{src}})} = \mathcal{X}$, a new behavior module B_{new} is generated if the conditions are satisfied:

(1) $\mathcal{B}$ does not include a behavior module B such that $LS^{(B)} = \mathsf{LS\text{-}accel}$ and $Src^{(B)} = B_{\mathrm{src}}$,

(2) $\overline{\sigma}_{B_{\mathrm{src}}} / \overline{\sigma}_{\max B_{\mathrm{src}}} < \sigma_{\mathrm{th}}$,

where $Src^{(B)}$ denotes the source behavior module of B, $\overline{\sigma}_B$ denotes a deviation of B's return, and $\overline{\sigma}_{\max B}$ denotes its maximum. Thus, the condition (2) checks if B_{src} almost converged, namely, is trained enough. σ_{th} is a threshold.

B_{new} uses the same BFs with B_{src}, namely $\mathcal{K}^{(B_{\mathrm{new}})} = \mathcal{K}^{(B_{\mathrm{src}})}$. The parameters of wire-fitting of B_{new} are copied from B_{src} except for $\{U_i^{(B_{\mathrm{new}})}\}$ that is multiplied by $\mathrm{f}_{\mathrm{accel}}$. Specifically, for each $i \in \mathcal{W}^{(B_{\mathrm{src}})}$,

$$\theta_i^{(B_{\mathrm{new}})} = \theta_i^{(B_{\mathrm{src}})}, \tag{5}$$

$$U_i^{(B_{\mathrm{new}})} = (g_i^{(B_{\mathrm{new}})}, q_i^{\mathrm{trg}(B_{\mathrm{new}})}) = (\mathrm{f}_{\mathrm{accel}} g_i^{(B_{\mathrm{src}})}, q_i^{\mathrm{trg}(B_{\mathrm{src}})}). \tag{6}$$

Additionally, the set of constraint range $\mathcal{I}_{\mathcal{R}} = \{(\mathrm{g}_i^{\mathrm{s}}, \mathrm{g}_i^{\mathrm{e}}) | i = 1, 2, \dots\}$ is also modified: $\mathrm{g}_i^{\mathrm{s}(B_{\mathrm{new}})} = \mathrm{f}_{\mathrm{accel}} \mathrm{g}_i^{\mathrm{s}(B_{\mathrm{src}})}, \mathrm{g}_i^{\mathrm{e}(B_{\mathrm{new}})} = \mathrm{f}_{\mathrm{accel}} \mathrm{g}_i^{\mathrm{e}(B_{\mathrm{src}})}$. Thus, the LS-accelerating transfers not only the policy parameters, but also the limitation of the policy.

3.3 LS-Freeing

The LS-freeing frees the DoF of a source behavior module to larger DoF based on a predefined freeing direction F. Each freeing direction F includes information, $\tilde{\mathcal{U}}_{\mathrm{src}}^{(F)}$, $\mathcal{X}_{\mathrm{src}}^{(F)}$, $\tilde{\mathcal{U}}_{\mathrm{dest}}^{(F)}$, and $\mathcal{X}_{\mathrm{dest}}^{(F)}$ which denote the spaces of a source behavior module and the spaces of a destination behavior module. This LS works with only a command space.

In the function $\mathsf{GEN}_{\mathrm{bhv}}(\mathsf{LS\text{-}free}, \tilde{\mathcal{U}}, \mathcal{X}, \mathit{Task})$, for each pair of (F, B_{src}) such that $\tilde{\mathcal{U}}_{\mathrm{dest}}^{(F)} = \tilde{\mathcal{U}}$, $\mathcal{X}_{\mathrm{dest}}^{(F)} = \mathcal{X}$, $\mathcal{U}^{(B_{\mathrm{src}})} = \tilde{\mathcal{U}}_{\mathrm{src}}^{(F)}$, and $\mathcal{X}^{(B_{\mathrm{src}})} = \mathcal{X}_{\mathrm{src}}^{(F)}$, a new behavior module B_{new} is generated if the conditions are satisfied:

(1) $\mathcal{B}$ does not include a behavior module B such that $LS^{(B)} = \mathsf{LS\text{-}free}$, $\tilde{\mathcal{U}}^{(B)} = \tilde{\mathcal{U}}$, $\mathcal{X}^{(B)} = \mathcal{X}$, and $Src^{(B)} = B_{\mathrm{src}}$,

(2) The same as the condition (2) of the LS-accelerating.

The condition (1) is to prevent to generate with the same setup.

B_{new} is initialized so that its action value function is almost the same as that of B_{src}. To do this, first, the freeing matrices $D_{\tilde{\mathcal{U}}}$ and $D_{\mathcal{X}}$ are calculated so that the conversions $\tilde{u}_{\mathrm{dest}} = D_{\tilde{\mathcal{U}}} \tilde{u}_{\mathrm{src}}$, $x_{\mathrm{dest}} = D_{\mathcal{X}} x_{\mathrm{src}}$ are performed where $\tilde{u}_{\mathrm{dest}} \in \tilde{\mathcal{U}}_{\mathrm{dest}}^{(F)}$, $\tilde{u}_{\mathrm{src}} \in \tilde{\mathcal{U}}_{\mathrm{src}}^{(F)}$, $x_{\mathrm{dest}} \in \mathcal{X}_{\mathrm{dest}}^{(F)}$, and $x_{\mathrm{src}} \in \mathcal{X}_{\mathrm{src}}^{(F)}$. We define these matrices as

$$D_{\tilde{\mathcal{U}}} = \mathrm{C}^{\sharp}_{\tilde{\mathcal{U}}_{\mathrm{dest}}^{(F)}} \mathrm{C}_{\tilde{\mathcal{U}}_{\mathrm{src}}^{(F)}}, \quad D_{\mathcal{X}} = \mathrm{C}_{\mathcal{X}_{\mathrm{dest}}^{(F)}} \mathrm{C}^{\sharp}_{\mathcal{X}_{\mathrm{src}}^{(F)}}, \tag{7}$$

where $\sharp$ denotes a pseudo-inverse. The parameters of the B_{new}'s BFs are calculated as

$$\mu_k^{(B_{\mathrm{new}})} = D_{\mathcal{X}} \mu_k^{(B_{\mathrm{src}})}, \quad \Sigma_k^{(B_{\mathrm{new}})} = D_{\mathcal{X}} \Sigma_k^{(B_{\mathrm{src}})} D_{\mathcal{X}}^{\top}, \tag{8}$$

for each $k \in \mathcal{K}^{(B_{\mathrm{src}})}$. The parameters of wire-fitting are initialized as

$$\theta_i^{(B_{\mathrm{new}})} = \theta_i^{(B_{\mathrm{src}})}, \tag{9}$$

$$U_i^{(B_{\mathrm{new}})} = (g_i^{(B_{\mathrm{new}})}, q_i^{\mathrm{trg}(B_{\mathrm{new}})}) = (g_i^{(B_{\mathrm{src}})}, D_{\tilde{\mathcal{U}}} q_i^{\mathrm{trg}(B_{\mathrm{src}})}), \tag{10}$$

for each $i \in \mathcal{W}^{(B_{\mathrm{src}})}$. In this case, $\mathcal{I}_{\mathcal{R}}$ is not changed.

3.4 LS-Planning

The function $\mathsf{GEN}_{\mathrm{bhv}}(\mathsf{LS\text{-}pln}, \mathcal{U}, \mathcal{X}, \mathit{Task})$ generates a behavior module if $\mathcal{U}$ is a discrete action space. Also, the generation with the same setup is prevented. A new behavior module uses Dyna-MG [6] where we use Q(λ)-learning instead of Q(0)-learning. Here, a default set of BFs is employed to linearly approximate the value function, and the reward and the dnyamics models.

3.5 LS-Hierarchy

The function $\mathsf{GEN}_{\mathrm{spl}}(\mathsf{LS\text{-}hier}, \mathit{Task})$ generates a hierarchical action space $\mathcal{H}$. The LS-hierarchy assumes that each task has a category label. A hierarchical action space is generated as a set of subtasks that have a same category label. Thus, the LS-hierarchy does not construct a hierarchical action space in a fully automatic manner. When an action in $\mathcal{H}$ is selected, the LS fusion algorithm is also used to execute the subtask. If a subtask is a continuing task (the HumanoidML-crawling and the HumanoidML-turning tasks are the case), we need to configure a duration of the subtask.

4. Experiments – HumanoidMaze

LS fusion is applied to a maze task of the simulated humanoid robot. In this task, the robot learns from scratch, namely, it learns not only a path to goal, but also a crawling and a turning motions. This task is referred to as HumanoidMaze. In order to train the robot, we specify a task sequence; 0-th to 1499-th episode: the crawling task, 1500-th to 2499-th episode: the turning-left task, 2500-th to 3499-th episode: the turning-right task, and 3500-th to 3999-th episode: the maze task.

In this learning, each LS is expected to be used in the following scenario:

(1) The primitive tasks (crawling and turning) are learned with the LS-scratch, the LS-freeing, and the LS-accelerating.

(2) The LS-hierarchy generates a hierarchical action space in which the primitive tasks are treated as the subtasks.

(3) The LS-planning generates a behavior module for the maze task where the hierarchical action space is used.

4.1 Task Setup

Fig. 2 shows the simulation model of the robot. Its height is 0.328m. It weights 1.20kg. Each joint torque is limited to 1.03Nm, and a PD-controller is embedded on it. The following experiments are performed on a dynamics simulator,

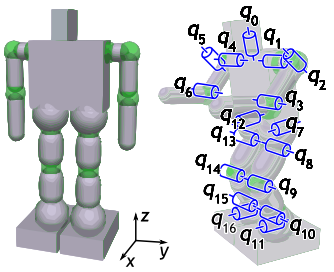

Fig. 2 Simulation model of a humanoid robot.

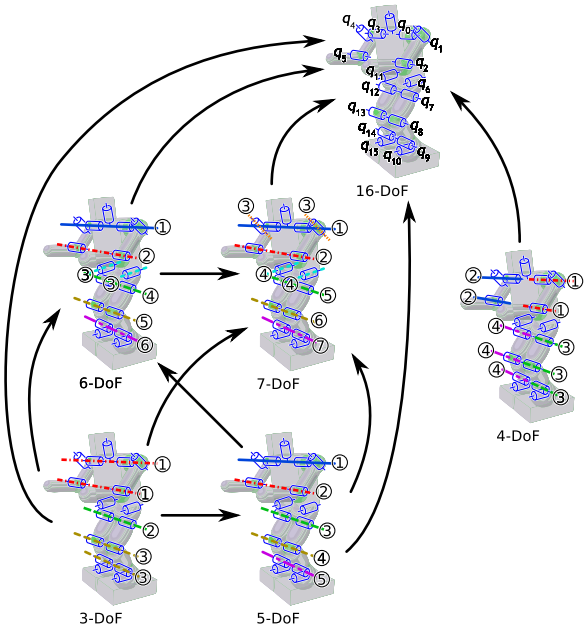

Fig. 3 DoF configurations and possible freeing directions. Each encircled number shows an index of dimension; joints with the same number are coupled. Each arrow shows that freeing is possible in this direction.

ODE(*3), with a time step 0.2ms. LS fusion is implemented with the RL library, SkyAI(*4).

We use six sets of DoF configurations (Fig. 3): 3-DoF $(\tilde{\mathcal{U}}_3, \mathcal{X}_3)$, 4-DoF $(\tilde{\mathcal{U}}_4, \mathcal{X}_4)$, 5-DoF $(\tilde{\mathcal{U}}_5, \mathcal{X}_5)$, 6-DoF $(\tilde{\mathcal{U}}_6, \mathcal{X}_6)$, 7-DoF $(\tilde{\mathcal{U}}_7, \mathcal{X}_7)$, and 16-DoF $(\tilde{\mathcal{U}}_{16}, \mathcal{X}_{16})$. In 3, 5, 6, and 7-DoF configurations, some joints are coupled to be bilaterally symmetric. In the 4-DoF, each leg has one DoF. In the 16-DoF, only the head link is fixed, while the other joints move independently.

The possible freeing directions between these DoF configurations are defined as shown in Fig. 3. Each arrow shows that freeing is possible in this direction.

The objective of the crawling task is to move forward as fast as possible. The objective of the turning task is to turn around the z-axis as fast as possible. In the both tasks, each episode begins with the initial state where the robot lies down and stationary, ends if $t > 20$[s] or the sum of reward is less than -40, which is caused by falling-down penalty. The objective of the maze task is to find a path from a start to a goal. When the robot reaches the goal, reward 10 is given and the episode is terminated.

(*3) : Open Dynamics Engine: `www.ode.org`

(*4) : SkyAI: `skyai.sourceforge.net`

4.2 Learning Method Configurations

We choose the parameters of LS fusion (denoted as LSF in the experiments) as follows: $\mathrm{f_{UCB}} = 2$, $\alpha_{\mathrm{R}} = 0.05$, $\mathrm{N_{LSBh}} = 10$, $\mathrm{N_{LSSp}} = 20$, $\tau_{\mathrm{lsd0}} = 20$, $\delta_{\tau_{\mathrm{lsd}}} = 0.004$, $\sigma_{\mathrm{th}} = 0.2$, and $\mathrm{f_{accel}} = 0.95$. The LS-scratch generates a behavior module whose parameters are set as $\tau_0 = 2$, $\delta_\tau = 0.02$ for Boltzmann selection of the decreasing temperature parameter $\tau = \tau_0 \exp(-\delta_\tau N_{\mathrm{eps}B})$ where $N_{\mathrm{eps}B}$ denotes a number of episodes performed by the behavior module B. The parameters of WF-DCOB are as follows: $C_{\mathrm{p}}(x) = \mathbf{q}_{N_{\mathrm{D}}}$, $C_{\mathrm{d}}(x) = \dot{\mathbf{q}}_{N_{\mathrm{D}}}$ for $x \in \mathcal{X}_{N_{\mathrm{D}}}$, and $\mathcal{I}_{\mathcal{R}} = \{(0.05, 0.1), (0.1, 0.2), (0.2, 0.3)\}$ for every configurations. The LS-accelerating and the LS-freeing generate a behavior module of the parameters $\tau_0 = 0.1$, $\delta_\tau = 0.02$ since the behavior module starts from an almost converged policy. Every behavior modules use Peng's Q(λ)-learning with $\gamma = 0.9$, $\lambda = 0.9$, and a decreasing step size parameter $\alpha = \min(0.05, 0.3 \exp(-0.002 N_{\mathrm{eps}B}))$.

4.3 Results

We execute 10 runs. Fig. 4 shows the learning curves of the first five runs of LSF (ex0,...,ex4). In the learning stage of the crawling and the turning tasks, we can find that the scenario (1) is achieved. Fig. 5 shows a learning curve and a behavior module transition in a run obtained in ex0 of Fig. 4. This graph shows how the set of behavior modules increases.

Fig. 6 shows the resulting learning curves of the task (the mean of the return per episode over 10 runs). In learning the maze task, one out of ten runs fails to acquire a path to goal. The major factor is considered to be a poor connection between the crawling policy and the turning policy. Using such policies as low-level actions may remove the Markov property from the maze task. Thus, the robot fails in the maze task. Such a failure will be avoided by mixing the primitive tasks in the primitive learning stage. This difficulty is not specific to LS fusion, but may arise in the other hierarchical RL methods. Thus, this failure does not dismiss the claim that LS fusion has a scalability to complex tasks.

Fig. 7 shows snapshots of an acquired behavior at the end of the maze task. We can see that the robot moves from start to goal by using the crawling and the turning motions.

5. Conclusion

This paper proposed the learning strategy (LS) fusion method where some LSs are integrated for learning a single task by a single robot. As the LSs, we defined the LS-scratch, the LS-accelerating, the LS-freeing, the LS-planning, and the LS-hierarchy. In the LS fusion algorithm, upper confidence bound (UCB) and Boltzmann selection method are employed to decide when and which LS is applied.

The proposed LS fusion was verified in a maze task of the simulated humanoid robot. In this task, the robot learned not only a path to goal, but also a crawling and a turning motions.

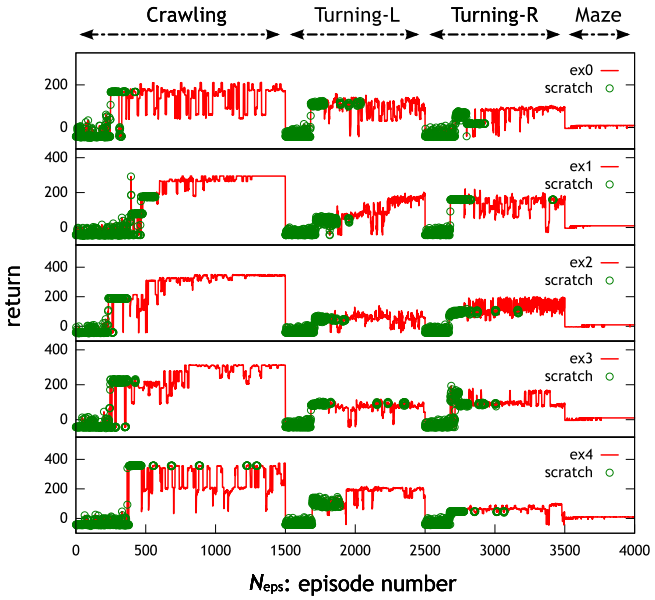

Fig. 4 Learning curves of five runs obtained from LSF. Each solid line shows the return per episode, and each circle shows the return acquired by a behavior generated by the LS-scratch.

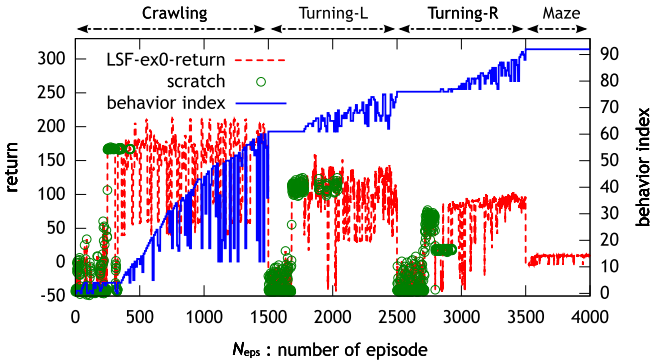

Fig. 5 Learning curve and module transition in a run obtained from LSF (ex0 in Fig. 4). The dotted line shows the return per episode, each circle shows the return acquired by a behavior generated by the LS-scratch, and the solid line shows the index of the selected behavior module in each episode.

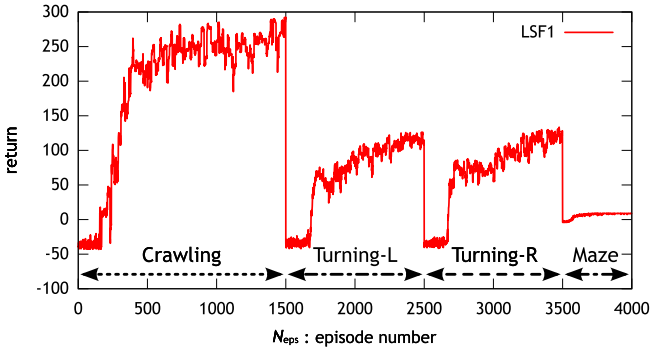

Fig. 6 Resulting learning curves of the HumanoidMaze task where the task changes at 1500, 2500, 3500-th episode. Each curve shows the mean of the return over 10 runs per episode.

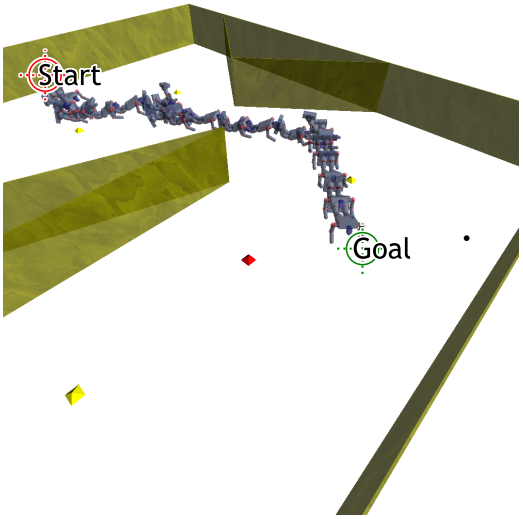

Fig. 7 Snapshots of an acquired behavior at the end of the HumanoidMaze task (taken in 1-FPS).

Our LS fusion architecture has a wide applicability; introducing the other LSs, such as using importance sampling technique [8] and imitation learning, is our future work.

Acknowledgements Part of this work was supported by a Grant-in-Aid for JSPS, Japan Society for the Promotion of Science, Fellows (22·9030).

Bibliography

[1] A. Yamaguchi, J. Takamatsu and T. Ogasawara: "Constructing continuous action space from basis functions for fast and stable reinforcement learning", the 18th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN'09), Toyama, Japan, pp. 401–407 (2009).

[2] J. Morimoto, S. Hyon, C. Atkeson and G. Cheng: "Low-dimensional feature extraction for humanoid locomotion using kernel dimension reduction", the IEEE Internactional Conference in Robotics and Automation (ICRA'08), pp. 2711–2716 (2008).

[3] Y. Takahashi and M. Asada: "Multi-layered learning systems for vision-based behavior acquisition of a real mobile robot", Proceedings of SICE Annual Conference 2003, pp. 2937–2942 (2003).

[4] J. Kober and J. Peters: "Learning motor primitives for robotics", the IEEE Internactional Conference in Robotics and Automation (ICRA'09), pp. 2509–2515 (2009).

[5] L. Torrey and J. Shavlik: "Transfer learning", Handbook of Research on Machine Learning Applications (Eds. by E. Soria, J. Martin, R. Magdalena, M. Martinez and A. Serrano), IGI Global, chapter 11 (2009).

[6] R. S. Sutton, C. Szepesvári, A. Geramifard and M. Bowling: "Dyna-style planning with linear function approximation and prioritized sweeping", Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence, pp. 528–536 (2008).

[7] L. C. Baird and A. H. Klopf: "Reinforcement learning with high-dimensional, continuous actions", Technical Report WL-TR-93-1147, Wright Laboratory, Wright-Patterson Air Force Base (1993).

[8] E. Uchibe and K. Doya: "Competitive-cooperative-concurrent reinforcement learning with importance sampling", In Proc. of International Conference on Simulation of Adaptive Behavior: From Animals and Animats, pp. 287–296 (2004).