Reward Shaping with Human Subgoals

by

Takato Okudo

Dissertation

submitted to the Department of Informatics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

The Graduate University for Advanced Studies, SOKENDAI

March 2023

A dissertation submitted to
Department of Informatics, School of Multidisciplinary Sciences,
The Graduate University for Advanced Studies, SOKENDAI,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Advisory Committee

1. Professor Seiji YAMADA
   National Institute of Informatics
   and The Graduate University for Advanced Studies, SOKENDAI

2. Associate Professor Tetsunari INAMURA
   National Institute of Informatics,
   and The Graduate University for Advanced Studies, SOKENDAI

3. Associate Professor Mahito SUGIYAMA
   National Institute of Informatics
   and The Graduate University for Advanced Studies, SOKENDAI

4. Professor Sachiyo ARAI
   Chiba University

5. Professor Tomohiro YAMAGUCHI
   National Institue of Technology(KOSEN), Nara College

Acknowledgements

I would like to express my deepest appreciation to my advisor Professor Seiji Yamada for his guidance, patience, and profound vision regarding the research. It has been a great honor to be his Ph.D. student. Without his continuous encouragement and contagious optimism, I could not have reached this far.

I would like to thank my dissertation committee, Associate Professor Tetsunari Inamura, Associate Professor Mahito Sugiyama, Professor Sachiyo Arai, and Professor Tomohiro Yamaguchi, for their constructive suggestions and invaluable comments, helping me to improve the research.

I would thank Dr. Song Sichao, Dr. Kazuo Okamura, Dr. Ryo Nakahashi, Shin Sano, Yosuke Fukuchi, Takahiro Tsumura, Nungduk Yun, Chenlin Hang, Jingbo Yan, Sota Kaneko, and the other members of Yamada Laboratory. I greatly appreciate their friendship as well as the good advice and comments on my research.

Sincere thanks go to Dr. Norikatsu Nagino, Director, Representative Executive Officer, and President of SCALA, who encouraged me to continue pursuing a doctoral degree.

Finally, very special thanks are dedicated to my family for all of the support they showed me during this research. I would like to thank my wife, Asuka, who has made countless sacrifices to help me get through this challenging time in the most positive way. This dissertation would not have been possible without her warm love, continued patience, and endless support.

The Graduate University for Advanced Studies, SOKENDAI

Abstract

School of Multidisciplinary Sciences
Department of Informatics

Doctor of Philosophy

Reward Shaping with Human Subgoals

by Takato Okudo

Many researchers have actively studied reinforcement learning, which acquires a policy maximizing long-term rewards. Unfortunately, this learning type needs to be faster and easier to use in practical situations because the state-action space becomes enormous in the real world. Many studies have incorporated human knowledge into reinforcement learning. Human knowledge of trajectories is common, but a human could be asked to control an AI agent. Controlling an AI agent could be too hard in specific tasks like robotics. Knowledge of subgoals may lessen this requirement because humans only need to consider a few representative states on an optimal trajectory. The essential factor for learning efficiency is rewards. Potential-based reward shaping is a primary method for enriching rewards and realizes a policy-invariant reward transformation that remains the optimal policy for an original reward function. A potential function is a real-valued function given a state, and the difference between its output in the current state and one in the previous state becomes a shaping reward. However, incorporating subgoals for accelerating learning over potential-based reward shaping is often challenging because the appropriate potentials are not intuitive for humans. We propose *subgoal-based reward shaping* based on potential-based reward shaping. Subgoal-based reward shaping includes a potential function given a state history and time. We prove that subgoal-based reward shaping is policy-invariant.

Subgoal-based reward shaping makes it easier for human trainers to share their knowledge of subgoals. Since the potential function is essential to make learning efficient, we propose two types of potential function, the *static goal-oriented potential* and *learned potential*, for subgoal-based reward shaping. The static goal-oriented potential approximates an optimal value function because the current study indicates that potential-based reward shaping makes policy learning efficient when the potential function is the optimal value function. We define a hyperparameter for the static goal-oriented potential that controls the shape of the potential. The result of an evaluation indicates that the

hyperparameter deteriorates the learning efficiency when inappropriate. To overcome this challenge, the learned potential acquires its potential simultaneously with the policy learning to remove the hyperparameter. We adopt a value function over abstract states that updates with n-step temporal difference (TD) learning during policy learning as a potential function for the learned potential. The abstract state is a subgoal achievement and begins from underachievement, and the transition in the abstract state follows the order of subgoals.

We conducted a user study to collect subgoal sequences from participants. The subgoals acquired from participants were more biased than random-generated subgoals, and many participants provided the same or similar subgoals.

We conducted simulation experiments in three domains covering discrete and continuous states and actions. The experimental results indicate the effectiveness with which subgoal-based reward shaping makes several baseline reinforcement learning algorithms, including a deep reinforcement learning algorithm, efficient. The learned potential achieves similar performance to the static goal-oriented potential. The results also indicate that the participants' subgoal sequences were superior to random-generated subgoal sequences for subgoal-based reward shaping.

A performance analysis between participants' and random subgoal sequences shows that the performances in domains where the subgoals were on optimal trajectories were similar. The best subgoal sequence of the participants was a part of more optimal trajectories than others. We found that an appropriate number of subgoals and a subgoal on an optimal trajectory can improve the baseline algorithm. Subgoal-based reward shaping performs well with a partially ordered subgoal sequence. The static goal-oriented potential is sensitive to changes in the hyperparameter, and initializing the potential improves the performance of the learned potential. Subgoal-based reward shaping cannot improve the baseline algorithm for negative step rewards. The learned potential is better than the static goal-oriented potential for mixed positive and negative rewards.

This dissertation does not propose an easy way to collect subgoal sequences from humans, but subgoal-based reward shaping can be applied to many domains as long as the user has subgoals. Though negative step rewards inhibit the effectiveness of subgoal-based reward shaping, they can be converted to a positive goal reward, for which subgoal-based reward shaping works well. Subgoal-based reward shaping can improve baseline reinforcement learning algorithms when a subgoal sequence is on an optimal trajectory. A detailed methodology might be optional, and it is enough for a subgoal teacher to have an optimal trajectory and subgoal sequence. This dissertation might encourage people to use subgoals yet to be used and help accelerate reinforcement learning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the topic of this dissertation and provides an overview of it. Section 1 introduces the challenge of reinforcement learning and our purpose and solutions, and Section 1.1 presents our contributions. Section 1.2 gives the structure of the dissertation. Reinforcement learning (RL) can acquire a policy maximizing long-term rewards in an environment. Designers do not need to specify how to achieve a goal; they only need to specify what a learning agent should achieve with a reward function. A reinforcement learning agent performs exploration and exploitation to determine how to achieve a goal. It is typical for the state-action space to be large in the real world. As the space becomes large, the number of iterations to learn the optimal policies exponentially increases, and the learning becomes too slow to obtain the optimal policies in a realistic amount of time. Since a human could have knowledge that would be helpful to such an agent in some cases, a promising approach is utilizing human knowledge [1]. Wang and Taylor's approach transfers a policy by requesting that a human select an action for a state during the learning phase [2]. Griffith et al. [3] used interactive human feedback as direct policy labels. This dissertation focuses on a more straightforward single-agent rather than a multi-agent.

The reward function is most related to learning efficiency. Most difficult tasks in RL have a sparse reward function [4]. The agent cannot evaluate its policy due to it and learn the optimal policy. In contrast, learning speeds up when the reward function is dense. Inverse reinforcement learning (IRL) [5] is the most popular method for enriching the reward function. IRL uses an optimal policy to generate a dense reward function.

Recent studies have utilized optimal trajectories [6]. There is the question of the cost for the teacher in providing the optimal trajectories or policies. Humans sometimes have difficulty providing these because of the skills they may or may not have. In particular, humans must have robot-handling skills and knowledge on the optimal trajectory in robotics tasks. We focus on the knowledge of subgoals. We consider a subgoal to be an intermediate goal to achieve a goal, and the subgoal provided by a human is often a part of the optimal trajectory. The subgoal in the field of RL is generally a state. The trajectory is a collection of time series tuples of a state, an action, and the next state. When the teacher provides subgoals, the teacher does not manage information on actions because the subgoals are a collection of states. This is why the knowledge of subgoals requires no robot-handling skills. Humans need only to think of a few states on the optimal trajectory. Moreover, though trajectories that fail are hard for IRL to use, subgoals can utilize them as a source. We used a reward-shaping approach to incorporate the knowledge of subgoals into RL. A general reward shaping makes reinforcement learning algorithms dirty because it varies the reward function, which affects other components such as the policy, actions, observations, and values. Potential-based reward shaping can add external rewards while keeping the optimal policy of the environment [7]. This characteristic of reward shaping is called policy invariance. It calculates an external reward on the basis of the difference between the real-number function (potential function) in the previous state and one in the current state. We call the output of a potential function a potential. Generally, a naive idea of the potential function for utilizing the knowledge of subgoals is high potential only for subgoals. However, as shown in Section 6, the potential function cannot perform well, so it is challenging to incorporate the knowledge of subgoals over potential-based reward shaping.

We propose subgoal-based reward shaping, which can generate additional rewards with a subgoal sequence on the basis of potential-based reward shaping. We adopt a history of states and time as the arguments of the potential function, and calculating an additional reward follows potential-based reward shaping. Judging whether a subgoal can be achieved requires a history of states in calculating a potential. We prove that subgoal-based reward shaping also satisfies policy invariance. A potential function with a subgoal sequence is essential to improve reinforcement learning in terms of learning efficiency. We first propose a potential function called the *static goal-oriented potential*, which increases its potential per subgoal achievement and keeps its potential between subgoal achievements. The experimental result of Ng et al. [7], which showed that potential-based reward shaping accelerates policy learning drastically when the potential function is an optimal value function, inspires us to design a static goal-oriented potential that approximates an optimal value function. The static goal-oriented potential requires the user to choose a hyperparameter controlling how much its potential increases per task.

The hyperparameter is time-consuming to choose because the user needs to estimate and search for it with technical knowledge. We second propose a potential function called the *learned potential*, which acquires its potential through interaction with an environment to remove the hyperparameter. The learned potential uses a value function over abstract states as the potential function and updates the value function with n-step TD learning. It almost follows SARSA-RS, which uses a value function over aggregated states as the potential function. The user provides a state aggregation function before learning, and SARSA-RS builds a value function over an abstract state space as the potential function. The propagation of the reward over the abstract state space is faster than over the original state space due to its small size, and the agent learns the policy faster. Devlin and Kudenko proved that potential-based reward shaping with a time-varied potential function whose arguments are state and time is policy-invariant [8]. The result has clarified that a policy learned with SARSA-RS is equivalent to the original policy since the time-varied potential function covers the value function over abstract states. However, it is not easy to define an aggregation function of states when the task has a high-dimensional state space. We propose dynamic state aggregation, the most central component of the learned potential, which aggregates the states an RL agent visits between subgoal achievements into an abstract state with a subgoal sequence. The aggregation enables the user to define a subgoal identification function instead of a state aggregation function. Since it is easier for the user to make a similarity function than an aggregation function, as shown in [9], our method can make the user effort lower than in the case of SARSA-RS. Moreover, a non-expert user may enhance the reinforcement learning algorithm if an identification function exists because the user provides only a subgoal sequence. User-friendliness might be related to interactive machine learning[10].

1.1 Contributions

The main contributions of this dissertation are:

- A user study that shows many subgoals provided by participants, and the distribution of the subgoals is biased more than random-generated subgoals.
- We prove policy invariance in an extension of dynamic potential-based reward shaping by adding an argument for state history. We propose the static goal-oriented potential over the theorem. The potential makes subgoal knowledge available to accelerate reinforcement learning.
- We propose a learned potential that removes a hyperparameter of the static goal-oriented potential and acquires the potential from learning. The user does not

need to tune the hyperparameter and can improve reinforcement learning with the learned potential.

- The experimental results show that the participants' subgoals were more useful for subgoal-based reward shaping than random-generated subgoals. A subgoal sequence over an optimal trajectory can accelerate reinforcement learning.

1.2 Outline

We organized this dissertation as follows:

Chapter 2 Related Work This chapter reviews existing research literature on human knowledge for reinforcement learning, utilization of subgoals, and reward shaping and clarifies the positioning of our methods.

Chapter 3 Background This chapter describes the basic methodology and methods for our methods, including reinforcement learning, reward shaping, and subgoals.

Chapter 4 Subgoal-based Reward Shaping This chapter proposes subgoal-based reward shaping. The proposal consists of an extension of dynamic potential-based reward shaping and two types of potential functions.

Chapter 5 Collecting Subgoal Provided by Human In this chapter, we present a user study for collecting subgoal sequences from participants and discuss them in three domains.

Chapter 6 Experiments for Evaluation This chapter evaluates our potential functions in terms of learning efficiency in every phase of learning and compares human-provided subgoal sequences with random-generated subgoal sequences per domain.

Chapter 7 General Discussion This chapter includes discussions about the dependency of subgoals, hyperparameters, types of rewards, and limitations for subgoal-based reward shaping.

Chapter 8 Conclusion This chapter summarizes our proposed methods and highlights the contributions of this dissertation.

Chapter 2

Related Work

This chapter gives the related work on human knowledge for reinforcement learning to figure out the positioning of this dissertation.

2.1 Human Knowledge for Reinforcement Learning

There are many studies utilizing human knowledge to RL; trajectory, policy, preference, action, feedback, and subgoal. From the policy or trajectory which human provides, IRL infers an unobserved reward function [11]. It is known to be difficult to design the reward functions because of an unspecified reward [12] and a reward hacking [13]. The unspecified reward is leaving out important aspects of reward design. The reward hacking is short of reward design for penalizing all possible misbehavior. Since the designer only provides the policy or trajectories without defining the reward function directly, the IRL overcomes these difficulties. The reward function is modeled as a linear sum of weighted features and is acquired by a margin-based optimization [5] or an entropy-based optimization [6]. Since the IRL generates a rich reward function, it is also a powerful solution when an environment has a sparse reward. The approach is the same as ours. The difference is that we use only some specific states as subgoals.

In RL with human preferences [14], a human teacher compares a pair of trajectory segments and selects the better one than the other one. The method learns a policy to get a high output of the human preference predictor without access to environmental

rewards. This is because preference learning assumes that following human preferences is optimal.

DAGGER [15] is an iterative algorithm that trains a deterministic policy under its induced distribution of states. DAGGER repeats the phases of collecting a dataset under the current policy and training the policy for the following phase. Human experts give labels for states that an agent visits in the collecting phase.

In the field of interactive reinforcement learning, a learning agent interacts with a human trainer as well as the environment[16]. The TAMER framework is a typical interactive framework for reinforcement learning[17]. The human trainer observes the agent's actions and provides binary feedback during learning. Since humans often do not have programming skills and knowledge of algorithms, the method relaxes the requirements to be a trainer. We aim for fewer trainer requirements, and we use a GUI on a web system in experiments with navigation tasks.

2.2 Utilization of Subgoals for Reinforcement Learning

Hierarchical and goal-conditioned RL mainly uses subgoals in the field of RL. Hierarchical RL(HRL) utilizes subgoals. The *option* framework is major in the field of HRL. The framework of Sutton et al. [18] could transfer learned policies in an option. An option consists of an initiation set, an intra-option policy, and a termination function. An option expresses a combination of a subtask and a policy for it. The termination function takes on the role of subgoal because it terminates an option and triggers the switching to another option. Recent methods have found good subgoals for a learner simultaneously with policy learning [19, 20]. The differences with our method are whether the policy is over abstract states or not and whether rewards are generated. The framework intends to recycle a learned policy, but our method focuses on improving learning efficiency.

The *RGoal* architecture by Ichisugi et al. [21] deals with recursive subgoals in HRL. The method solves an MDP with an augmented state space in which each subgoal has a state space. The method speeds up learning in multi-task settings by sharing subroutines between tasks. The objective differs from ours, and the paper suggests that a non-hierarchical RL method is faster than HRL in single-task settings. Murata et al. [22] used subgoals to accelerate RL learning. A reward is generated upon the achievement of each subgoal, and there is a Q function for each subgoal. The policy selects an action according to the weighted sum of Q functions of subgoals and an original Q function. This method applies only to table-style value functions. Our method can be applied to function-style value functions.

Goal-conditioned RL extends RL towards tackling multiple goals simultaneously [23]. In goal-conditioned RL, the goals which an agent should achieve are given. The simple way is to extend the value function by adding a goal called *universal value function* [24]. Nair et al. [25] proposed Reinforcement Learning with Imagined Goals (RIG). RIG learns broadly applicable and general-purpose skill repertoires for an agent to fulfill user-specific goals. Nasiriany et al. [26] incorporated goal-conditioned RL into planning. They introduced *feasivility vector*, which evaluates how each planning is feasible for achieving each subgoal. They minimized the norm of the feasibility vector to acquire good planning. Our method focuses on the standard RL problem. Hindsight experience replay (HER) algorithm [27] regards a failure as a success in hindsight, and it generates a reward for the failure. The algorithm interprets a failure as a process for achieving a target, similar to a subgoal. HER accelerates learning in a multi-goal environment. We focus on a single-goal environment, but we evaluate learned potential with random subgoals, which is similar to HER.

The other study investigates how the representation of a subgoal can be acquired in high-dimensional observation. Siyuan et al. [28] proposed a method that extracted features with slow dynamics from observations as the subgoal spaces. The learned features highlighted the invariant observation in the short term. We abstract states by the achievement status of subgoals.

There are a few methods compared with our method. For a fair comparison, the method's objective and input, which is a subgoal series, should be the same as ours.

2.3 Reward Shaping

Marom and Rosman [29] proposed reward shaping based on Bayesian methods called belief reward shaping(BRS). The expected value over the estimated probability distribution of an environmental reward over a state and action is used as a shaping reward. They proved that the policy learned by Q-learning augmented with BRS is consistent with an optimal policy over the original MDP.

The landmark-based reward shaping of Demir et al. [30] is the closest to our method. The method shapes reward only on a landmark using a value function. Their study focused on a POMDP environment, and landmarks automatically become abstract states. We focus on a Markov decision process (MDP) environment and acquire subgoals from human participants, and we apply our method to a task with high-dimensional observations.

Reward shaping in HRL has been studied in [31, 32]. Gao et al. [31] showed that potential-based reward shaping remains policy invariant to the MAX-Q algorithm. Designing potentials at every level is laborious work. We use a single high-level value function as a potential, which reduces the design load. Li et al. [32] incorporated an advantage function in high-level state-action space into reward shaping. The reward shaping method in [33] utilized subgoals automatically discovered from expert trajectories. The potential generated at each subgoal is different.

Potential-based advice is reward shaping for states and actions[34]. The method shapes the Q-value function directly for a state and an action, making it easy for a human to advise an agent whether an action in an arbitrary state is better than the others. Subgoals show what ought to be achieved on the trajectory to a goal. We adopted the shaping of a state value function.

Harutyunyan et al. [35] have shown that the Q-values learned by arbitrary rewards can be used for potential-based advice. The method mainly assumes that a teacher negates the agent's selected action. The method uses failures in trial and error. In contrast, our method uses success.

Grzes and Kudenko[36] analyzed the effectiveness of two types of potential in a single goal reward and penalized step rewards. They proposed the requirements of a potential function to make learning efficient. The potential function should motivate a transition to a high potential state, and it should penalize staying in the same state and moving back to the previous state. We focus here on the analysis of their work if the discount factor is lower than 1, $\gamma < 1$. The positive potentials were worse than the negative in the step penalized rewards, and the negative potentials could improve learning, but it got worse when it significantly violated the requirements. Laud and DeJong [37] analyzed the reward shaping with a concept of *reward horizon*. They stated that easy-to-learn MDPs have low reward horizons, and an excellent reward-shaping method lowers the reward horizon.

Chapter 3

Background

This chapter provides the background of this dissertation. Section 3.1 introduces reinforcement learning, and section 3.2 presents reward shaping. They are the basic frameworks of our dissertation. Section 3.3 reviews the existing studies regarding subgoal.

3.1 Reinforcement Learning

Reinforcement learning is a framework to acquire a policy maximizing future rewards through interactions between an agent and an environment. Reinforcement learning works under a Markov decision process (MDP). An MDP is represented as (S, A, T, γ, R), where S is a finite set of states, A is a set of actions of $k \geq 2$, $T = Pr(s_{t+1}|s_t, a)$ is a state transition function, $\gamma \in (0, 1]$ is the discount factor, and R specifies a reward function. An agent has a policy $\pi(a|s)$ and a value function $V(s)$, $Q(s, a)$. Two ways can be used to learn the optimal policy, a value-based method [38] and a policy-gradient-based method [39]. The value-based method indirectly learns a policy by estimating the optimal value function. The policy gradient method-based method directly learns the parameter of a policy. We explain the value-based method, Q-learning [40], in this section. In a learning experiment, we used a hybrid method of value-based and policy gradient called the actor-critic algorithm [41].

The initial state is $s_0 \in S$. The agent selects action $a \in A$ in the current state s_t, then the environment returns the next state s_{t+1} and reward r. The agent updates a policy

$\pi(a|s)$ based on reward r. Note that S is a state space and that A is an action space. The value function is defined as follows,

$$V_\pi(s_t) = \mathbb{E}_\pi\left[\sum_{i=t}^{\infty}\gamma^{i-1}r_i|s_t = s_t\right] \tag{3.1}$$

$$Q_\pi(s_t, a) = \mathbb{E}_\pi\left[\sum_{i=t}^{\infty}\gamma^{i-1}r_i|s_t = s_t, a = a\right] \tag{3.2}$$

The above equations (3.1) and (3.2) show an expected discounted cumulative reward following the policy π when an agent is on a state or a state-action. γ is a discount rate, V is a value function and Q is an action-value function(Q function). The optimal values are shown as follows,

$$\begin{aligned} V^*_\pi(s_t) &= \max_\pi V_\pi(s_t) \\ &= \max_{a'} Q^*(s_t, a') \\ Q^*_\pi(s_t, a) &= \max_\pi Q_\pi(s_t, a) \\ &= r_t + V^*(s_{t+1}) = r_t + \max_{a'} Q^*(s_{t+1}, a') \end{aligned}$$

The agent does not know the dynamics of an environment like a state transition function and reward function. The agent needs to estimate them by sampling. Q-learning [40] is a core reinforcement learning method to estimate the optimal action-value function. The updated equation is

$$Q_\pi(s_t, a) \leftarrow Q_\pi(s_t, a) + \alpha\left(r + \gamma \max_{a'} Q_\pi(s_{t+1}, a') - Q_\pi(s_t, a)\right) \tag{3.3}$$

α is the learning rate. The action value function converges to the optimal one under $0 < \alpha < 1$ and an infinite number of trials. The rest of this section explains the RL algorithm which we employed as a baseline in Chapter6.

3.1.1 SARSA

SARSA is one of the most basic value-based RL algorithms. SARSA stands for State, Action, Reward, State, and Action, which means that it uses them when the agent updates its value. The update rule is written formally as follows,

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a)) \tag{3.4}$$

α is a learning rate, s and a are a current state and action, s' and a' are a next state and action, and r is a reward. The equation is similar to Q-learning. The target is only different from Q-learning. The next state s' is selected by the agent's policy π, and SARSA is on-policy learning.

3.1.2 Actor Critic

Actor-critic methods are TD methods that have a separate memory structure to represent the policy independent of the value function. The policy structure is *actor*, and the estimated value function is *critic*. The critic criticizes the actions decided by the actor. Learning is always on-policy. The critique takes the form of a TD error. Actor-critic methods have variations. We describe what we employed in Chapter 6. We employed Q-function as a critic and Gibbs softmax method as an actor. Actions are generated by the following probability distribution.

$$\pi_\theta(a|s) = P\{A = a|S = s\} = \frac{\exp H_\theta(s,a)}{\sum_b \exp H_\theta(s,b)} \tag{3.5}$$

α is a learning rate. This distribution outputs the action more frequently as its action has more H_θ. H is a modifiable function with the policy parameters θ of the actor. The parameter updates by following the policy gradient theorem in Equation 3.6 [40].

$$\theta \leftarrow \theta + \alpha Q(s,a) \nabla_\theta \ln \pi_\theta(a|s) \tag{3.6}$$

α is a learning rate. SARSA updates the Q-function of the critic.

3.1.3 Deep Deterministic Policy Gradient

DDPG stands for Deep Deterministic Policy Gradient and is a model-free off-policy actor-critic algorithm [42]. DDPG extends DQN [43] to continuous action space with the actor-critic method while learning a deterministic policy. DQN uses a deep neural network as a Q-value function, and the Q-value function becomes non-linear. Q-learning may suffer from instability and divergence when combined with a non-linear Q-value function approximation and bootstrapping. DQN employed two mechanisms, experience replay and periodically updated target to stabilize the training procedure. The experience replay stores the experiences the agent acquires in a replay memory and samples the experiences from the memory at random when its policy and value update. Experience replay removes correlations in the observation sequences, and smooths over

changes in the data distribution. Since the periodically updated target is not used in DDPG, we omit the description. The loss function looks as follows.

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2\right] \tag{3.7}$$

where $U(D)$ is a uniform distribution over the replay memory D, θ^- is the parameters of the frozen target Q-network. DDPG also employs these two mechanisms. DDPG trains the actor with the deterministic policy gradient [44].

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^\mu}[\nabla_a Q^\mu(s, a)\nabla_\theta \mu_\theta(s)|_{a=\mu_\theta(s)}] \tag{3.8}$$

ρ^μ is a discounted state distribution, defined as

$$\rho^\mu(s') = \int_{\mathcal{S}} \sum_{k=1}^{\infty} \gamma^{k-1} \rho_0(s)\rho^\mu(s \to s', k)ds \tag{3.9}$$

$\rho_0(s)$ is the initial distribution over states, and $\rho^\mu(s \to s', k)$ is the visitation probability density at state s' after moving k steps by deterministic policy μ. We employed the exploration policy constructed by adding noise $\mathcal{N}$.

$$\mu'(s) = \mu_\theta(s) + \mathcal{N} \tag{3.10}$$

DDPG does soft updates on the parameters of actor and critic, with $\tau \ll 1 : \theta' \leftarrow \theta + (1 - \tau)\theta'$. It makes the target network values constrained to change slowly instead of the periodically updated target of DQN

3.2 Reward Shaping

3.2.1 Potential-based Reward Shaping

Potential-based reward shaping (PBRS) [7, 45] is an effective method for keeping an original optimal policy in an environment with an additional reward function. If the potential-based shaping function F is formed as

$$F(s_t, s_{t+1}) = \gamma\Phi(s_{t+1}) - \Phi(s_t) \tag{3.11}$$

it is guaranteed that policies in MDP $M = (S, A, T, \gamma, R)$ are consistent with those in MDP $M' = (S, A, T, \gamma, R + F)$. $\Phi(s_t)$ is a function with a state s_t as an argument. Wiewiora showed that a reinforcement learner with initial Q-values based on the potential function makes the same updates throughout learning as a learner receiving potential-based shaping reward [45]. We assume two learners, L and L'. L initializes the Q-value to $Q_0(s, a)$ and updates its policy with an environmental reward r and the shaping reward F. The Q-value of L' is initialized to $Q'_0(s, a) = Q_0(s, a) + \Phi(s)$, and L' uses only the reward r. The Q-values are disassembled into the initial values Q_0 and Q'_0 and the updated values $\Delta Q(s, a)$ and $\Delta Q'(s, a)$ through learning, respectively. If L and L' learn on the same sequence of experiences, $\Delta Q(s, a)$ is always equal to $\Delta Q'(s, a)$. Moreover, value-based policies of L and L' have an identical probability distribution for their next action. Our proposed method uses potential-based shaping rewards to keep an optimal policy in an original MDP.

3.2.2 Dynamic Potential-based Reward Shaping

Devlin and Kudenko [8] have shown that the extension of potential-based reward shaping with time series also remains the optimal policy. They called the method dynamic potential-based reward shaping. It is written in 3.12 formally.

$$F(s, t, s', t') = \gamma\Phi(s', t') - \Phi(s, t) \tag{3.12}$$

s' is a state observed at the current time t', and s is at the previous time t. The potential $\Phi(s, t)$ covers the potential varying with time. To prove policy invariance they showed that the return a shaped agent will receive for following a fixed sequence of states and actions is equal to the return the non-shaped agent would receive when following the same sequence minus the potential of the first state in the sequence. We employed this strategy to prove policy invariance. Though potential-based reward shaping is equivalent to Q-table initialization, dynamic potential-based is not equivalent because of the time-varied potential. The time-varied Q value cannot be provided in the initialization phase. Their empirical studies with a single agent showed that their potential function took longer to learn the optimal policy than no reward shaping. They discussed that their potential function was a uniform random and the function had no specific knowledge. They also showed the empirical result on multi-agents. The dynamic potential function made the optimal policy easy to learn. We focused on an environment with a single agent, and show our dynamic potential function took shorter to learn the optimal policy than no reward shaping.

3.2.3 SARSA-RS

Grzes et al. [46, 47] proposed a method that learns a potential function Φ while learning a policy π, called "SARSA-RS". The method solved the problem of designing an appropriate potential function for a difficult and time-consuming domain. We define Z as a set of abstract states. The method builds a value function over Z and uses it as Φ

$$\Phi(s) = V(g(s)) = V(z) \tag{3.13}$$

, where g is an aggregation function, $g : S \to Z$. The function g is pre-defined. The potential-based shaping function over SARSA-RS is written as follows.

$$F(z_t, z_{t+1}) = \gamma V(z_{t+1}) - V(z_t) \tag{3.14}$$

The method learns the value function $V(z)$ during policy learning as

$$V(z_t) \leftarrow V(z_t) + \alpha \left(r_h + \gamma^k V(z_{t+1}) - V(z_t) \right) \tag{3.15}$$

, where r_h is the transformation function from MDP rewards into SMDP rewards, and k is the duration between z_t and z_{t+1}. The potential function changes dynamically during learning, and the equivalency of the potential-based reward shaping cannot be applied because it depends on the time and the state. Since Devlin and Kudenko have shown that a shaped policy is equivalent to a non-shaped one, when the potential function changes dynamically during learning[8], SARSA-RS keeps the learned policy original. We omit the time argument in the following section to simplify the expression. The size of Z is smaller than S thanks to the aggregation of states. Therefore, the propagation of environmental rewards is faster, and policy learning with SARSA-RS is also faster. As mentioned above, the method requires the pre-defined aggregation function g. In an environment of high-dimensional observations, it is almost impossible to make an aggregation function.

3.3 Subgoal

This section overviews a notion of subgoal. The subgoals are used in many kinds of research but are not defined uniformly. A subgoal in the field of reinforcement learning is classified into a state on more optimal trajectories [48–53] and a final state in the set of states with similar features [28, 33, 54]. Many methods automatically try to find the subgoal through learning. Their subgoal focused on the improvement of reinforcement learning and ease of computation. Hierarchical reinforcement learning (HRL) often

adopts a subgoal defined as a state on more optimal trajectories. Our methods make abstract state space with subgoals similar to HRL, so we referred to the definition.

A human learner can learn a mental model for solving a task more effectively if the task is divided into subgoals in the field of educational psychology. It is known as *subgoal learning*, and providing subgoals is called *subgoal labeling* [55]. Catrambone [56] defined a subgoal as the task structure to be learned for solving problems in a particular domain. Weir et al. [57] developed the workflow to acquire subgoal labels on *learnersourcing* which is a method for human computation for gathering information from people trying to actively learn from a video. They asked people the intention of a step or a set of steps with free-form textual input for subgoal labeling. We also employed free-form textual input to acquire subgoals from a video in Chapter 5.

Chapter 4

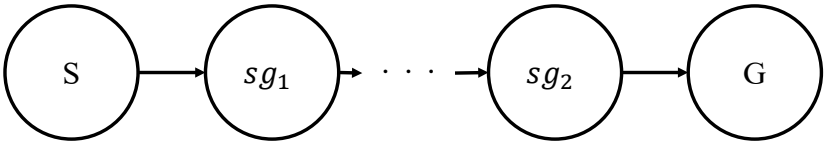
Subgoal-based Reward Shaping

In this chapter, we propose *subgoal-based reward shaping*. Section 4.1 defines the target problem for subgoal-based reward shaping, and Section 4.2 extends dynamic potential-based reward shaping to incorporate subgoals and proves the policy invariance of the extension. Section 4.3 proposes *static goal-oriented potential*, and Section 4.4 proposes the *learned potential* to make learning efficient in subgoal-based reward shaping. Section 4.5 finally presents the whole algorithm of subgoal-based reward shaping.

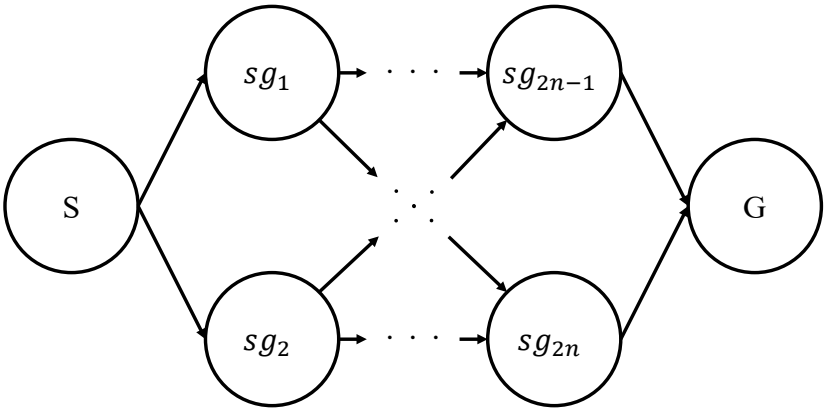
4.1 Problem Definition

We first define the target problem of subgoal-based reward shaping, which assumes MDP with a given sequence of subgoals. We denote a set of subgoals as SG and a sequence of subgoals as $(SG, \succ)$. The subgoals are a subset of a set of states $SG \subset S$. The sequence of subgoals can be both partially ordered and totally ordered. A totally ordered subgoal sequence deterministically determines the following subgoal sequence for any subgoal. In contrast, a partially ordered subgoal sequence branches to several following subgoal sequences for any subgoal. Figure 4.1 shows a totally ordered and partially ordered subgoal sequence.

In this figure, S is an initial state, G is a goal state, and $sg_n \in S$ denotes the n-th subgoal. Figure 4.1(B) shows two subgoal sequences, $[sg_1, \cdots, sg_{2n-1}]$ and $[sg_2, \cdots, sg_{2n}]$. If an agent is in S, it can select which subgoal sequences start from sg_1 and start from sg_2—



(A): Totally ordered subgoal sequence.



(B): Partially ordered subgoal sequence.

FIGURE 4.1: Totally ordered and partially ordered subgoal sequence.

the subgoal sequence branches after starting from S. An agent cannot achieve sg_1 after achieving sg_2. As an example of partially ordered subgoals, a whole subgoal sequence for Figure 4.1(B) can be described as $[[sg_1, \cdots, sg_{2n-1}], [sg_2, \cdots, sg_{2n}]]$. Our proposed method can handle both types of subgoals. MDP with a sequence of subgoals is written as a tuple $(S, A, T, \gamma, R, (SG, \succ))$. An agent interacts with an environment to learn its policy, and it can access the given sequence of subgoals. The shaping reward F outputs the difference of the potentials, the same as potential-based reward shaping. The agent utilizes the sum of the environmental reward and the shaping reward $R + F$. The sequence of subgoals is utilized only for calculating F in subgoal-based reward shaping. The RL algorithm, which works in MDP, is available because adding the sequence of subgoals only affects the shaping reward.

4.2 Subgoal-based Reward Shaping Guarantees Policy Invariance

We extended dynamic potential-based reward shaping to allow the history of the states an agent visited and the time as parameters of a potential function. Informally, if the potential function obtains the history of states in a difference equation, the guarantees of policy invariance remain. Formally,

$$F(h_j, h_{j+1}, t_j, t_{j+1}) = \gamma\Phi(h_{j+1}, t_{j+1}) - \Phi(h_j, t_j) \tag{4.1}$$

where h_j is the history of the states that an agent visited until state s_j at time t_j, expressed as an equation, $h_j = [s_0, \cdots, s_j]$. To prove the policy invariance, we first define the return in any arbitrary policy π when experiencing sequence ζ in a discount manner without shaping. Formally,

$$U(\zeta) = \sum_{j=0}^{\infty} \gamma^j r_j \tag{4.2}$$

Note that j is a time step. The return U rewrites the Q function as follows:

$$Q_\pi(s, a) = \sum_{\zeta} Pr(\zeta|s, a)U(\zeta) \tag{4.3}$$

Now consider the same policy but with a reward function modified by adding a potential-based reward function of the form given in Equation (4.1). The return of the shaped policy U_Φ experiencing the same sequence ζ is,

$$\begin{aligned}
U_\Phi(\zeta) &= \sum_{j=0}^{\infty} \gamma^j \left(r_j + F(h_j, h_{j+1}, t_j, t_{j+1})\right) \\
&= \sum_{j=0}^{\infty} \gamma^j \left(r_j + \gamma\Phi(h_{j+1}, t_{j+1}) - \Phi(h_j, t_j)\right) \\
&= \sum_{j=0}^{\infty} \gamma^j r_j + \sum_{j=0}^{\infty} \gamma^{j+1}\Phi(h_{j+1}, t_{j+1}) - \sum_{j=0}^{\infty} \gamma^j \Phi(h_j, t_j) \\
&= \sum_{j=0}^{\infty} \gamma^j r_j + \sum_{j=1}^{\infty} \gamma^j \Phi(h_j, t_j) - \sum_{j=1}^{\infty} \gamma^j \Phi(h_j, t_j) - \Phi(h_0, t_0) \\
&= \sum_{j=0}^{\infty} \gamma^j r_j - \Phi(h_0, t_0) \\
&= U(\zeta) - \Phi(h_0, t_0) = U(\zeta) - \Phi(s_0, t_0)
\end{aligned} \tag{4.4}$$

where $h_j = [s_0, \cdots, s_j]$ and $h_{j+1} = [s_0, \cdots, s_j, s_{j+1}]$ are written. Since $h_0 = [s_0]$, h_0 is equivalent to s_0. $Q_{\pi,\Phi}$ follows from Equation (4.3) and is expressed as

$$
\begin{aligned}
Q_{\pi,\Phi}(s,a) &= \sum_{\zeta} Pr(\zeta|s,a)U_{\Phi}(\zeta) \\
&= \sum_{\zeta} Pr(\zeta|s,a)\left(U(\zeta) - \Phi(s,t)\right) \\
&= \sum_{\zeta} Pr(\zeta|s,a)U(\zeta) - \sum_{\zeta} Pr(\zeta|s,a)\Phi(s,t) \\
&= Q_{\pi}(s,a) - \Phi(s,t) \qquad (4.5)
\end{aligned}
$$

Given by Equation (4.5), Φ does not have action a in parameters. As Φ is independent of the action taken, then in any given state, the best action remains constant regardless of the shaping. Therefore, we can conclude that the guarantee of policy invariance remains.

4.3 Static Goal-oriented Potential

The subgoal-based potential is the central part of our method. Since the potential can make learning significantly efficient if the potential is an optimal value, we design potentials that utilize a subgoal sequence to approximate the optimal values. Generally, the larger the optimal value is, the closer a state is to an environment with only a goal reward. We consider subgoal achievement as approaching the goal, and we design a potential that grows with every subgoal achievement. Formally, we define subgoal-based potentials as:

$$\Phi(h,\cdot) = \eta * c(h) \qquad (4.6)$$

where η is a hyperparameter, and $c(h)$ is a function for counting the number of subgoals achieved in the order of the given subgoal sequence in a visited state sequence of h. The potential function is independent of an argument of time. Figure 4.2 shows an output sequence of the potential function.

The upper horizontal axis is the time t, the lower is the event, and the vertical axis is the potential in Figure 4.2. An agent achieves three subgoals in order of the given subgoal sequence. When the agent achieves sg_1, the potential function generates $\eta * c(h)$. Upon achieving sg_2, this value becomes larger than sg_1 because $c(h)$ increases. The function continues the potential between subgoals, which is the crucial part of this potential function. The potential intuitively has a value only in subgoals if the designer incorporates the knowledge of subgoals into potential-based reward shaping, but the potential does not perform well, as shown in chapter 6. The static goal-oriented potential requires a

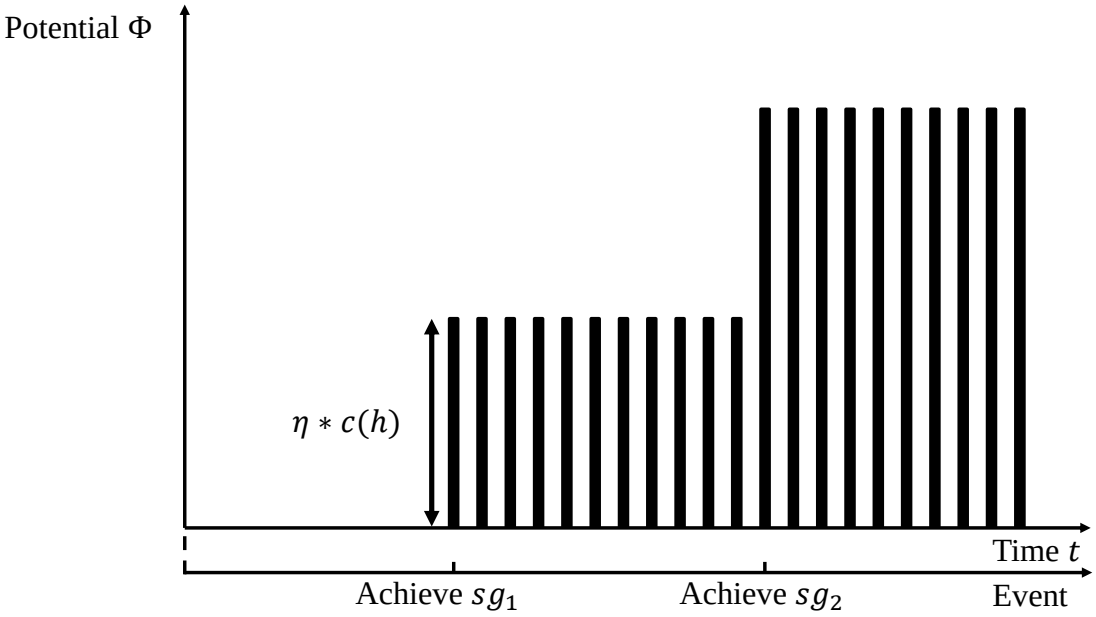


FIGURE 4.2: Potential function for subgoals.

hyperparameter η controlling how much its potential increases. It is time-consuming to determine the hyperparameter as shown in Section 7.2. We remove the hyperparameter from the equation and acquire it by learning. We can aggregate the states between subgoal achievements and build abstract states because the potentials between subgoal achievements are the same. The following section proposes the learned potential, similar to SARSA-RS, using a value function over abstract states as the potential.

4.4 Learned Potential with Dynamic State Aggregation

We propose a *learned potential*. The learned potential simultaneously learns its potential with the policy learning in reinforcement learning to remove the hyperparameter and acquire it from learning. It is based on SARSA-RS and uses an abstract state value as a potential. It is formally written as

$$\Phi(h,t) = V_t(g(h)) \tag{4.7}$$

$$z = g(h) \tag{4.8}$$

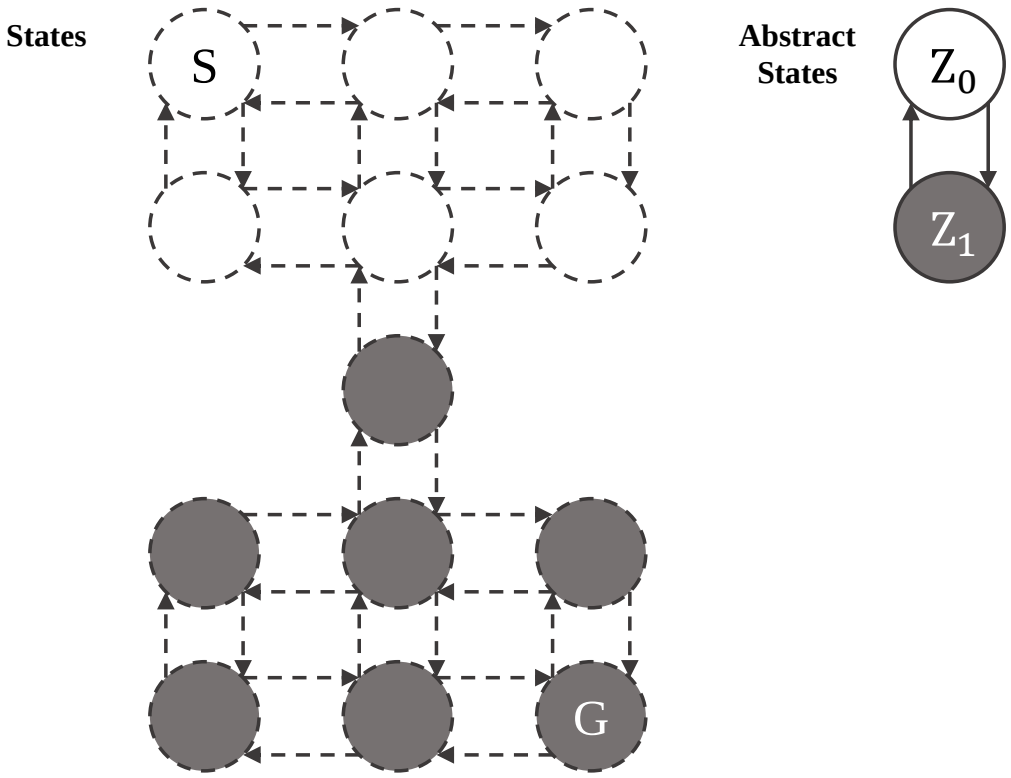
Equation 4.7 shows that the potential is a value over abstract states. $g(h)$ is a state aggregation function to an abstract state z. Since $\Phi(h,t)$ is guaranteed to be policy-invariant, the learned potential is also policy-invariant. The value function depends

on the time t because the learned potential updates the abstract state value. The aggregation function $g(h)$ dynamically aggregates into an abstract state using a subgoal sequence, called *dynamic state aggregation*. The method can make the SARSA-RS method available for environments of high-dimensional observations thanks to less effort being required from designers. The method requires only a subgoal sequence consisting of several states instead of all states for SARSA-RS. Section 4.4.1 explains the dynamic state aggregation. Section 4.4.2 presents how to update the abstract state value, and Section 4.4.3 introduces how the learned potential works with an experience replay.

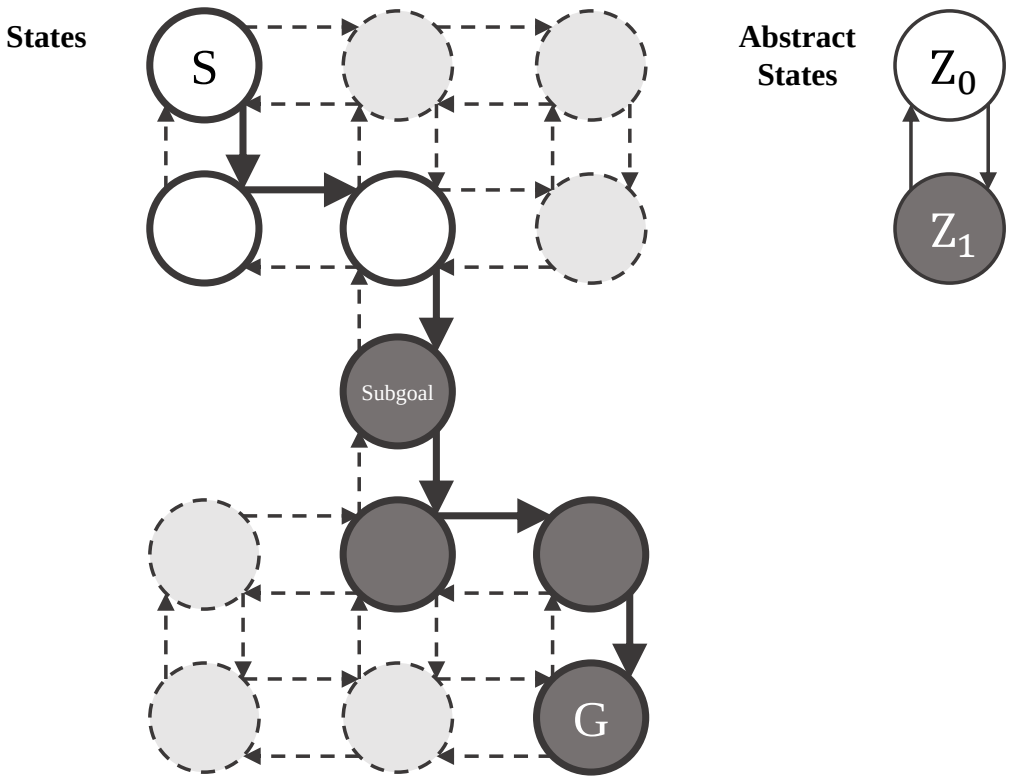
4.4.1 Dynamic State Aggregation

We build abstract states that represent the achievement status of a subgoal sequence. If there are n subgoals, the size of the abstract states is $n + 1$. The abstract states are one-directional because it is difficult to judge whether the achievement status has been returned. The agent is in a first abstract state z_0 before it achieves a subgoal. Then, the abstract state z_0 transits to z_1 when it achieves the subgoal sg_0. This means that the state aggregation, which includes episodes until subgoal sg_0, transits into z_0. The aggregated states change dynamically every trial because of the randomness of the policy and learning. A relationship between a state and an abstract state is not a one-to-one correspondence. As the learning progresses, the aggregated states gradually become fixed because the policy approximates an optimal one. The exploration strategy generally decreases the randomness of the policy. The shaping reward with learned potential distributes the value of an abstract state to the values of states included in the trajectory. Note that the trajectories for updating the values differ from those of the distributed values. The updated value function is not used for the current trial but for the subsequent trials. Figure 4.3 shows an image of dynamic state aggregation.

In the figure, a circle is a state or an abstract state, an arrow between circles is a transition, and the aggregated states are colored white or dark gray. A single subgoal makes two abstract states. The white circle of an abstract state aggregates the white circles of a state. “S” and “G” represent that the circles including them are a start and goal state, respectively. The bold circles and arrows express the trajectory that the agent generates. Figure 4.3(A) shows that the trajectory is separated into two sub-trajectories, each of which corresponds to the abstract states. The designer must provide all relationships between a state and an abstract state in Figure 4.3(A). In contrast, the dynamic state aggregation only requires subgoals as shown in Figure 4.3(B). This is why the designer does not need much effort.



(A): Aggregation function in SARSA-RS.



(B): Aggregation function in dynamic state aggregation.

FIGURE 4.3: Concept of dynamic state aggregation.

4.4.2 Abstract State Value Update

We use n-step TD learning [40] to update values over abstract states because we want the abstract state value to have the value of the state selected as a subgoal. An n-step backup is defined as a backup toward the n-step return, and the target of the n-step TD learning includes it. The target for an n-step backup might be

$$r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}) \tag{4.9}$$

where s_t, r_t, and V_t are the state, the reward, and the value function at time t, respectively. The state updates its value with the target in n-step TD learning. We only have an abstract state aggregating the states, and the target might be

$$r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(z') \tag{4.10}$$

where V is the abstract state's value, and z' is the next abstract state. Though the normal n-step TD learning needs to calculate every state, our setting requires the calculation only for the abstract state and fewer computations. N-step TD learning has an *error reduction property*, in which the worst error under the new estimate is guaranteed to be less than or equal to γ^n times the worst error under any value function $v\colon S \to \mathbb{R}$. An n-step return using v is guaranteed to be a better estimate than a one-step return using v. Finally, the equation of n-step TD learning in our setting for updating an abstract state value is

$$V(z) \leftarrow V(z) + \alpha \left(\sum_{i}^{n} \gamma^{i-1} r_{t+i} + \gamma^n V(z') - V(z) \right) \tag{4.11}$$

4.4.3 Potential Calculation in Experience Replay

This section describes an additional process for the learned potential when the baseline algorithm uses an experience replay method. An experience replay stores a set of experiences and samples the experiences by batch size when an agent updates its policy and value as mentioned in Section 3.1.3. It also stores a shaping reward in replay memory when the agent gets an experience. The learned potential continues to update its potential and to change the shaping reward. Therefore, the stored shaping reward might be updated after it is stored. Since the updated potential is generally richer, we want to generate the shaping reward from the updated potential when the agent learns. To realize this, we store a current abstract state and the following abstract state in

replay memory and calculate the shaping reward when sampling. The process makes the shaping reward the latest value, and the agent can utilize a richer shaping reward.

4.5 Whole Algorithm

The pseudocode of the static goal-oriented potential is shown in Algorithm 1:

Algorithm 1 Static goal-oriented potential.

Ensure: γ, η, SG

$s \leftarrow s_0, h \leftarrow [], h' \leftarrow []$
repeat
  Choose a according to π
  Take a in s, observe s', r
  $h' \leftarrow h$
  Append s' into h'
  $\Phi(h', \cdot) = \eta * c(h')$
  $F(h, \cdot, h', \cdot) = \gamma\Phi(h', \cdot) - \Phi(h, \cdot)$
  Update value function and policy with $r + F(h, \cdot, h', \cdot)$
  $s \leftarrow s', h \leftarrow h'$
until termination.

Though we consider a totally ordered subgoal sequence in the experiments, we can easily extend the algorithm with a partially ordered subgoal sequence. We modify only the function $c(h)$ for a partially ordered subgoal sequence. We discuss a partially ordered subgoal sequence in Section 7.3. After an agent takes action a and receives a state s' from an environment, the state s' is added to the history of states h'. The calculation of Φ involves h and h'. The function $\Phi(h)$ is used to compute $F(h, h')$. The algorithm can work even if we give only subgoals without a sequence. Since they are partially ordered subgoals with only a single tier of a sequence of subgoals, the extension mentioned above is available.

The pseudocode of the learned potential is shown in Algorithm 2:

In Algorithm 2, the method is involved between lines 6-13. The value function assumes a table form. If s' equals sg_{i+1}, V is updated by a multi-step TD method [40], and our method sets the next subgoal sg_{i+1} in lines 9-12. The totally ordered subgoal sequence determines the following subgoal automatically, but the partially ordered subgoal sequence does not. Section 7.3 reports the logic for a partially ordered subgoal sequence. The big difference from Algorithm 1 is the update part of the abstract value.

Algorithm 2 Learned potential.

Ensure: $t, V_0(z), sg_i \in \{SG, \prec\}$

$z \leftarrow z_0, i \leftarrow 0, r_h \leftarrow 0, h \leftarrow []$
Choose a according to π
repeat
    Take a in s, observe s', r
    Append s' into h
    $z' \leftarrow g(h)$
    $t \leftarrow t + 1, i \leftarrow i + 1$
    $r_h \leftarrow r_h + \gamma^i r$
    if $equal(s', sg_{i+1})$ || *done* **then**
        $\delta = r_h + \gamma_v^i V_t(z') - V_t(z)$
        $V_{t+1}(z) \leftarrow V_t(z) + \alpha_v \delta$
        $t \leftarrow 0, i \leftarrow 0, r_h \leftarrow 0$
    $F = \gamma V_{t+1}(z') - V_t(z)$
    Choose a' according to π
    Update value function and policy with $r + F$
    $s \leftarrow s'; a \leftarrow a', z \leftarrow z'$
until termination.

Chapter 5

Collecting Subgoal Sequence Provided by Human

In this chapter, we explain a user study done to acquire human knowledge of subgoals expressed by subgoal sequences. We used navigation tasks in two domains, four-room and pinball, that express the state space spatially. We also conducted a pick-and-place task, a primary robotics task for which humans can have difficulty controlling the robot. The source code for acquiring subgoal sequences is available on GitHub.[1]

5.1 Navigation Task

Navigation is when an agent tries to reach a goal position from a start position with the minimum number of steps. The agent can move on a plane. We prepared four-room and pinball domains. The agent can move in four directions in the four-room domain; it moves its ball in four directions in the pinball domain. The agent in the pinball domain can also not use force. Participants provided subgoal sequences online. We conducted an online user study to acquire human subgoal knowledge using the web-based GUI in Figure 5.1.

The blue square shows the start position, and the orange shows the goal position in Figure 5.1. The sidebar included the flow of the user study, and a part for interaction

[1] https://github.com/takato86/subgoal_experiment

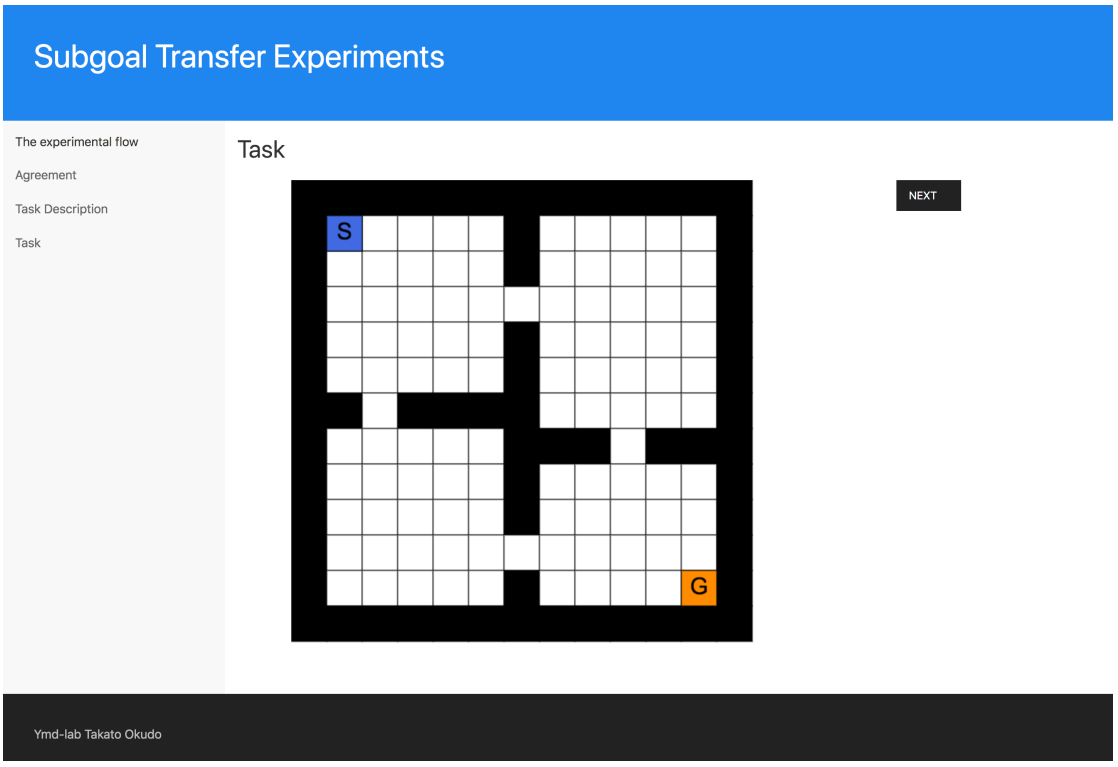


FIGURE 5.1: UI for acquiring subgoal sequence.

was placed in the center of the page. Figure 5.1 shows the four-room domain, and the pinball domain was located in the same place. The area in the interaction part was clickable and turned green when a user clicked on it. The user could press the next button and proceed to the next task. We recruited 10 participants, half graduate students in the department of computer science and half others (six males and four females, ages 23 to 60, average of 36.4). We confirmed they did not have expertise on subgoals in the two domains. We gave the participants the same instructions for the two domains and then asked them to designate their subgoal sequence, including two subgoals, in this fixed order for the four-room and pinball domains. The number of subgoals was the same as the hallways in the optimal trajectory for the four-room domain and was two. The instructions explained to the participants what a subgoal is and how to set one through the GUI shown in Figure 5.1. The instructions for subgoals stated, "Please state two positions that must be gone through to reach the goal in the shortest path and their orders." Also, specific explanations of the two task domains were given to the participants. In this experiment, we acquired just two subgoals for learning since they are intuitively considered easy to give on the basis of the structure of the problems. The given subgoal sequence was in total order.

5.2 Robotics Task

A user study for the pick-and-place task was also done online. Since acquiring human subgoal knowledge with a GUI was challenging, we used a descriptive answer-type form. We assumed that humans use subgoals when they verbally teach behavior. They state not how to move but what to achieve in the middle of behavior. The results of this paper slightly support this assumption. We recruited five participants who were amateurs in the field of computer science (three males and two females, ages 23 to 61, average of 38.4). The participants read the instructions and then typed the answer in a web form. Figure 5.2 shows the Google form for acquiring subgoal sequences of the pick-and-place task. The reader can try answering the Google form from the following link[2].

As shown in Figure 5.2, the instructions consisted of a description of the pick-and-place task, a movie about manipulator failures, a glossary, and a question on how a human can teach successful behavior. The question included the sentence "Please teach behavior like you would teach your child." This is because some participants answered that they did not know how to teach the robot in a preliminary experiment. We imposed no limit on the number of subgoals and the type of order.

5.3 Analysis of Participants' Subgoals

This section analyzed the participants' subgoals without their orders because every participant's subgoal sequence, in which a larger order is closer to the goal, was almost the same. We compared the participants' subgoals with random-generated subgoals in the four-room and pinball domains to determine whether the distribution was biased. We sampled all subgoals from NumPy's uniform distribution [58]. We also randomly decided on the order of subgoals to use them for subgoal-based reward shaping in Chapter 6. The population in the four-room domain is a state space, and in the pinball domain, it is a set of vectors on an x-y plane. Figure 5.3 shows the subgoal distribution acquired from the participants and the random-generated subgoal distribution in the four-room domain.

In Figure 5.3(A), the color of the start point, the goal, and the subgoals are blue, red, and green, respectively. The number in a cell in Figure 5.3 counted for the participants that selected that cell as a subgoal. As shown in Figure (A), many participants selected the hallways. Four participants selected the hallways on the bottom-left route, and two selected the hallways. Since there is no duplicate selection, the number in a cell equals the number of participants. The near-center cells in the top-right and bottom-right rooms

[2]https://forms.gle/iwZT2ynhVxzoPwQXA

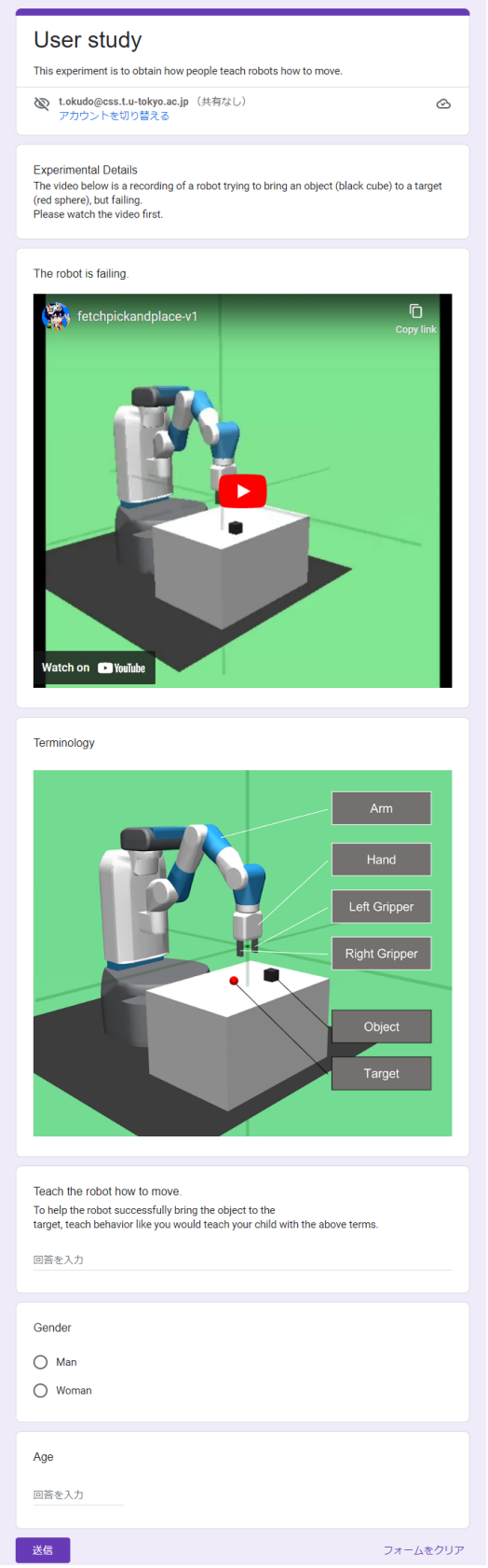
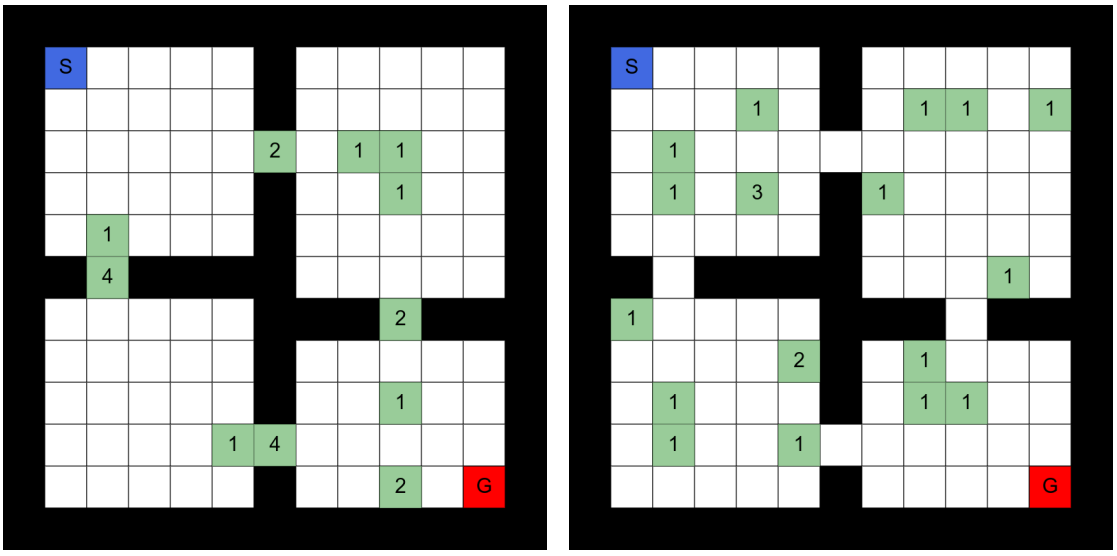
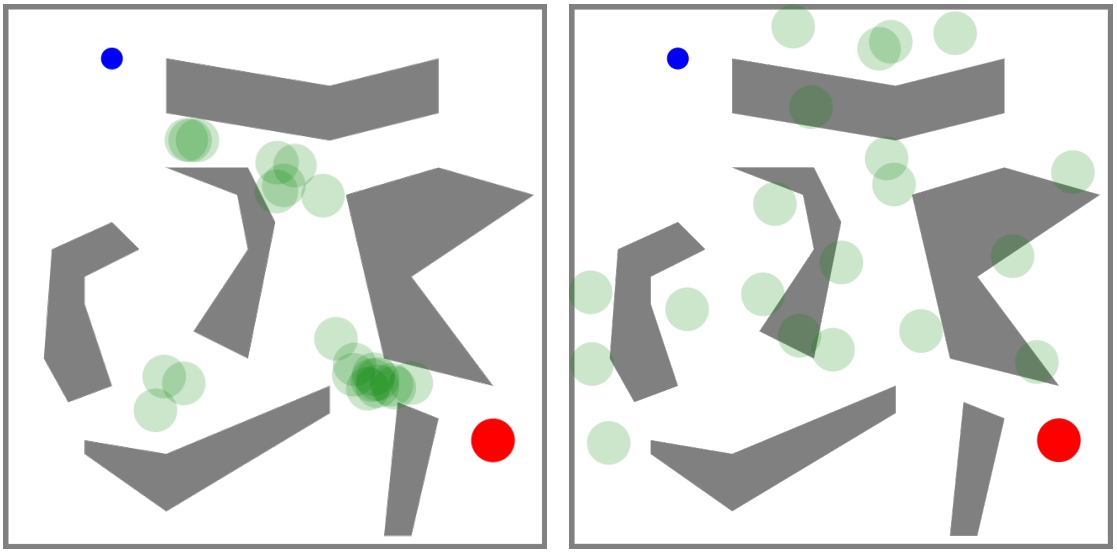


FIGURE 5.2: Google form for acquiring subgoals of pick-and-place task.



(A): Participants' subgoals. (B): Random subgoals.

FIGURE 5.3: Subgoal distributions of four-room domain.



(A): Participants' subgoals. (B): Random subgoals.

FIGURE 5.4: Subgoal distributions of pinball domain.

were selected. The participants equally selected the subgoals on the bottom-left and top-right routes. The random-generated subgoals were scattered in every room, as shown in Figure 5.3(B). The top-left room had six, the bottom-left had six, the top-right had five, and the bottom-right had three. Figure 5.4 shows the subgoal distribution acquired from the participants and the random subgoal distribution in the pinball domain.

A blue circle is a ball, a red circle is a goal area, and a green circle is a subgoal in Figure 5.4. The deep green means that many participants selected the position. We predefined the radius of the green circle to be the same as the goal but not the participant's decision. The participant could check the radius of the subgoal after a click. The

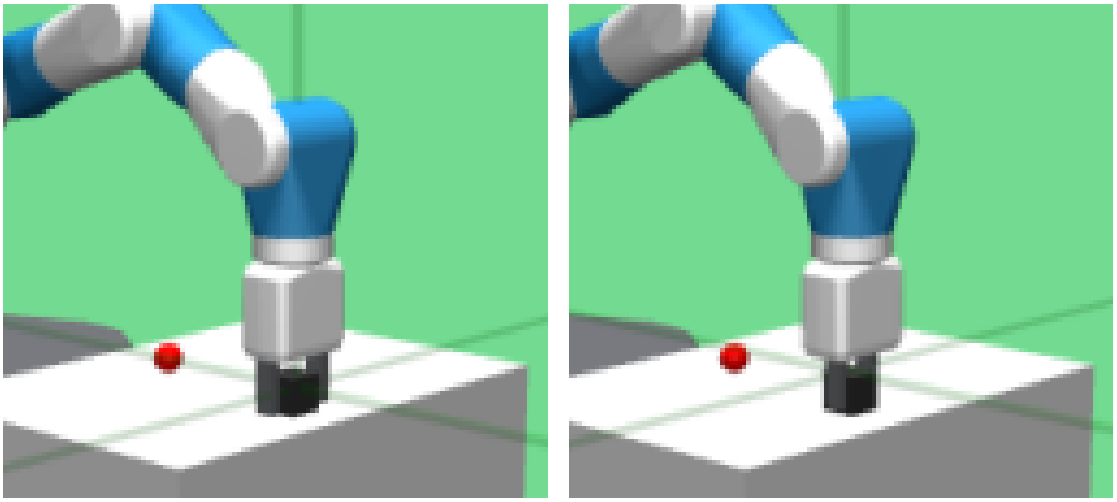
Table 5.1: List of participants' answers.

Participant	Answer
1	Align the hand's center with the object's center and pinch it between the left and right gripper. Then, move the arm so that the hand's center aligns with the target's center.
2	Align the center of the gripper with the object, place it between the right and left grippers, lift it, and carry it to the target.
3	Bring the center of the gripper directly above the object to catch it and bring it to the target.
4	First, check the current object position and target position. In this case, the object is located to the left of the left gripper. First, move the object to the left so that the center of the gripper is at the center of the object. When the center of the gripper is at the center of the object, grab the object. Next, move the gripped object so that the center of the target overlaps the center of the object. When the center of the target aligns with the center of the object, place the object.
5	When you have the center of the gripper over the object, lower the gripper toward the bottom, close the gripper, grab the object, lift the object, bring it to the top of the target, and then lower the gripper and release the object.

participants focused on four regions of branch points to set subgoals in comparison with random subgoals from Figure 5.4(A) and Figure 5.4(B). Most subgoals were placed on a branch point before the goal and did not cover obstacles. In contrast, some random-generated subgoals covered obstacles. Table 5.1 lists the participants' answers for the pick-and-place domain.

We picked a subgoal sequence including two subgoals from each participant's answer as listed in Table 5.1 to align the number of subgoals with the other domains. All the participants determined the same subgoals; the first subgoal involved reaching the location available to grasp the object, and the second subgoal involved grasping the object. We adopted them as subgoal sequences for subgoal-based reward shaping. Figure 5.5 visualizes the subgoal sequence in the pick-and-place domain.

A black cube is an object, and a red cube is a target position in Figure 5.5.



(A): First subgoal. (B): Second subgoal.

FIGURE 5.5: Visualization of subgoals from participants in pick-and-place domain.

Chapter 6

Experiment for Evaluation

This chapter presents how the proposed two potential functions work in three domains. Section 6.1 introduces the evaluation metrics, and section 6.2 explains the comparative methods to our methods. The sections show the experimental results in section 6.3 for the navigation task in the four-room domain, in section 6.4 for the navigation task in the pinball domain, and in section 6.5 for the pick and place task with a fetch robot. Section 6.6 summarizes the experimental results.

6.1 Evaluation Metric

We evaluated the learning efficiency of subgoal-based reward shaping with Jumpstart, Time to Threshold, and Asymptotic Performance [59]. Taylor and Stone [59] summarized five performance metrics used in the field of transfer learning for RL, i.e., Jumpstart, Asymptotic Performance, Total Reward, Transfer Ratio, and Time to Threshold. The setting of transfer learning has a source task and a target task. A transferring agent learns the source task and then transfers the knowledge to another agent. The agent with the transferred knowledge learns the target task with the knowledge. The above performance metrics are used to evaluate how much the learning efficiency improves with the knowledge transfer from the source to the target task. Since humans do many tasks in daily life, we assumed that humans learn the source task daily and transfer the knowledge to an agent in our task as the target task. This assumes that the tasks of daily life include tasks similar to ours. A sequence of subgoals is transferred knowledge in our

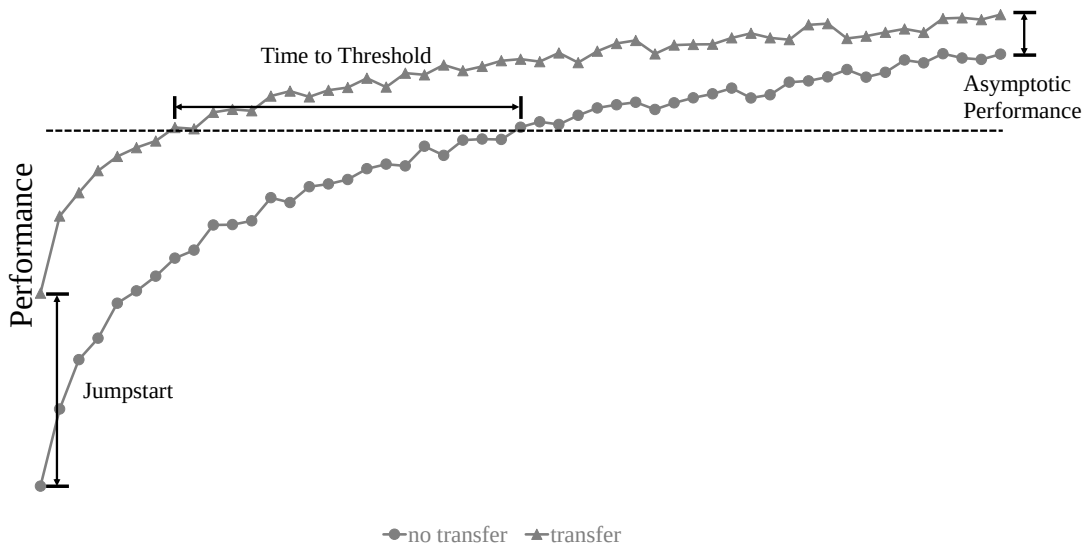


FIGURE 6.1: Comparisons in terms of performance metrics.

setting, and the performance metrics of transfer learning can be used to evaluate subgoal-based reward shaping. The performance metrics can also be used to evaluate the quality of the sequence of subgoals as the transferred knowledge if we fix the reward-shaping method. We selected Jumpstart, Time to Threshold, and Asymptotic Performance to measure the performance in the early, middle, and final phases of learning, respectively. Jumpstart is the initial performance in a target task. It shows how much a transfer from a source task improves in the early phase. Time to Threshold is the number of episodes to get below or above a predefined number of threshold steps, and it can measure in the middle phase. We set thresholds to 500, 300, 100, and 50 for the four-room domain, 3000, 2000, 1000, and 500 for the pinball domain, and 0.2, 0.4, 0.6, and 0.8 for the pick-and-place domain. We observed the learning curves and then determined the thresholds. Asymptotic Performance is the final learning performance and shows the performance in the final phase. All the metrics in pinball and pick-and-place domains were calculated after a simple moving average of ten episodes. Figure 6.1 shows the comparisons in terms of these performance metrics as a learning curve to make it easier to understand them.

We followed the evaluation by Taylor and Stone [60]. They conducted statistical tests with these metrics. Since we have two or more comparison algorithms, we applied analysis of variance (ANOVA) and then conducted multiple comparisons with Tukey's Honestly Significant Difference (HSD) method as post-doc tests. We analyzed the metrics in the statistical tests with js-STAR XR+ [61] version 1.3.0 j.

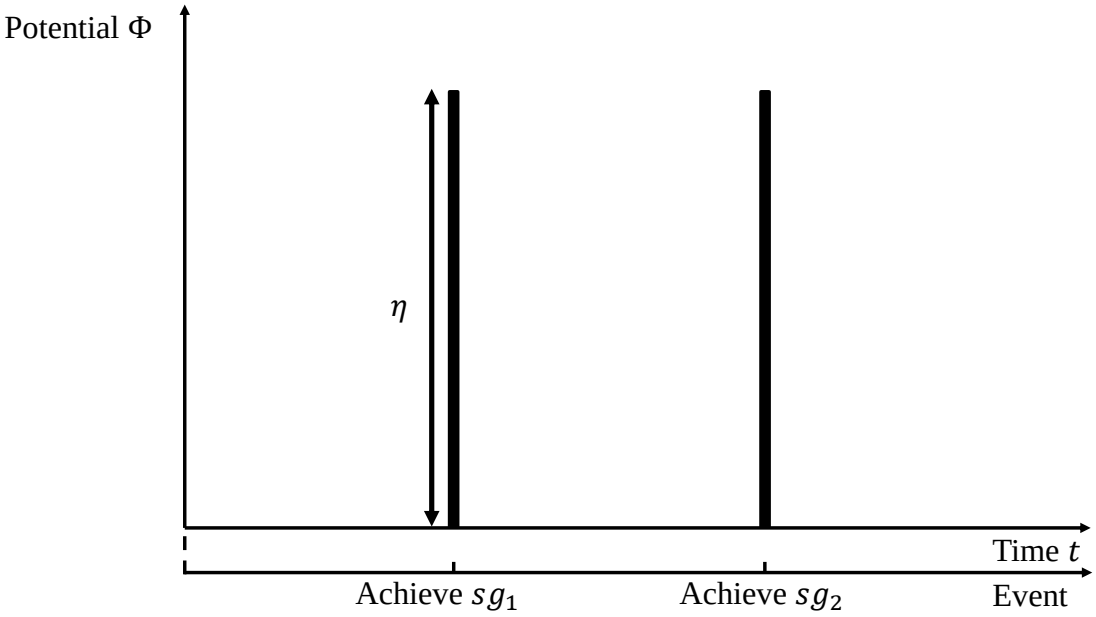


FIGURE 6.2: Potential function of NRS.

6.2 Comparison Algorithms

We compared learned potential (LEARNED) with several baseline RL algorithms and three other types of reward-shaping algorithms, i.e., subgoal-based reward shaping with static goal-oriented potential (STATIC), naive subgoal reward shaping (NRS) and linear naive subgoal reward shaping (LINRS). For a fair comparison, the algorithm needs to be the same as the input: a sequence of subgoals. NRS is a simple algorithm, and many algorithm designers should use it first. In NRS, the function $\Phi(s)$ in potential-based reward shaping generates a scalar value η when an agent visits a subgoal state; η is a hyperparameter. The potential function is written as follows:

$$\Phi(s) = \begin{cases} \eta & s = sg \\ 0 & s \neq sg \end{cases} \tag{6.1}$$

Figure 6.2 shows the potential function of NRS.

The axes of Figure 6.2 are the same as Figure 4.2. When the event occurs, the function outputs η. LINRS extends NRS by increasing its potential every step that the agent achieves the subgoal. The designer also utilizes this method commonly when the designer enriches the reward function. The potential function of LINRS is

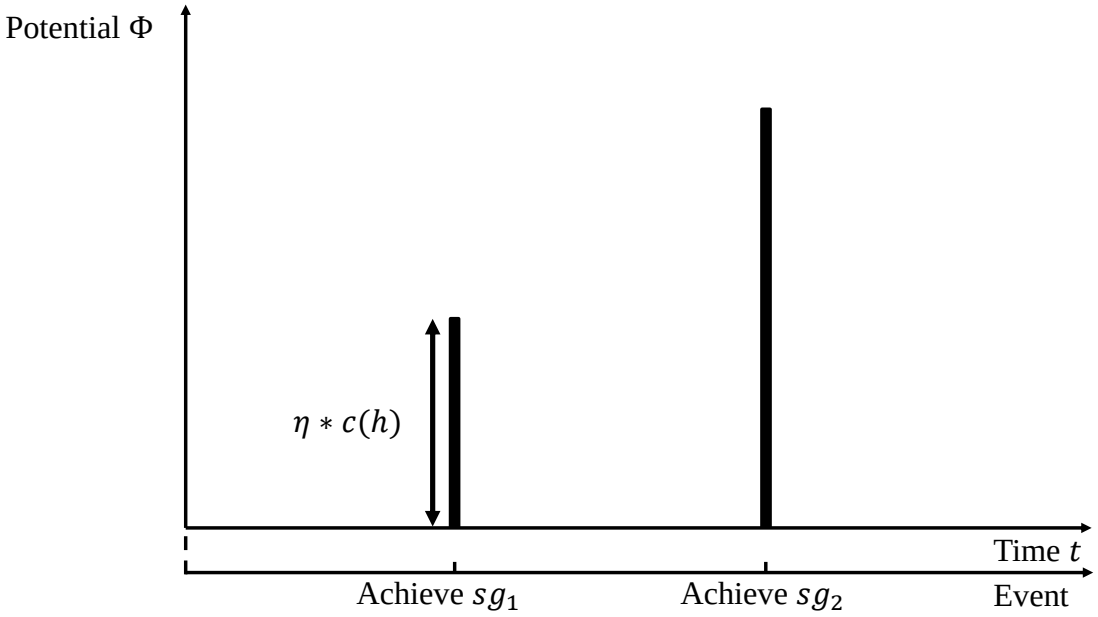


FIGURE 6.3: Potential function of LINRS.

$$\Phi(h) = \begin{cases} \eta * c(h) & s = sg \\ 0 & s \neq sg \end{cases} \tag{6.2}$$

Since counting achieved subgoals is necessary, the input of the potential function is a state history h. LINRS only differs from STATIC in that it does not keep the potential value and only outputs when the agent achieves the subgoal. Figure 6.3 shows the potential function of LINRS.

The axes of Figure 6.3 are the same as Figure 4.2. When the agent achieves sg_1, the $c(h)$ outputs one, and then the potential is η. The agent gets 2η in achieving sg_2 because $c(h)$ outputs two. Though the potential of NRS is the same value between sg_1 and sg_2, the potential of LINRS in sg_2 is bigger than sg_1. We implemented a different baseline RL algorithm for each domain to show that our methods (STATIC and LEARNED) can be available over various RL algorithms. All the comparison reward-shaping algorithms perform over the baseline RL algorithm. Table 6.1 shows the summary list of abbreviations. We also use the abbreviation in the rest of this paper.

Table 6.1: Summary list of abbreviations

Abbreviation	Explanation
SARSA	We explain SARSA in section 3.1.1.
AC	Actor-critic which we explained in section 3.1.2.
DDPG	We explain DDPG in section 3.1.3.
LEARNED	Learned potential which potential is dynamic.
STATIC	Static goal-oriented potential which potential is static.
NRS	Naive reward shaping
LINRS	Linear naive reward shaping
HUMAN	A subgoal sequence provided by each participant
RANDOM	A random-generated subgoal sequence

6.3 Navigation in Four-room Domain

6.3.1 Environmental Setup

The four-room domain, a common RL task, has four rooms and four hallways connecting the rooms. Learning consisted of 250 episodes. An episode was a trial run until an agent successfully reached a goal state or when 1000 state actions failed. A state was expressed as a scalar value labeled through all states. An agent could select an action from up, down, left, and right. The agent's action failed with a probability of 0.33 and then changed to another random action. A reward of +1 occurred when the agent reached a goal state and 0 otherwise. The start state and goal state was at a fixed position. The agent repeated learning 100 times.

6.3.2 Algorithmic Setup

We used SARSA, a basic RL algorithm, as the baseline for the four-room domain. The policy was soft-max with a table-formed q-value function. We set the learning rate to 0.01 and the discount rate to 0.99. The function $c(h)$ counted the number of subgoals in a sequence in order from the front of a state sequence h. The minimum output was zero, and the maximum was two. We set η to 0.1 after conducting grid-search on 0.01, 0.1, 1, 10, and 100. The results of the grid search were presented in section 7.2.

6.3.3 Utilization of Subgoals

The agent used each sequence of two subgoals the participant provided in learning. A subgoal for the four-room domain was the rectangle the participant pointed out because

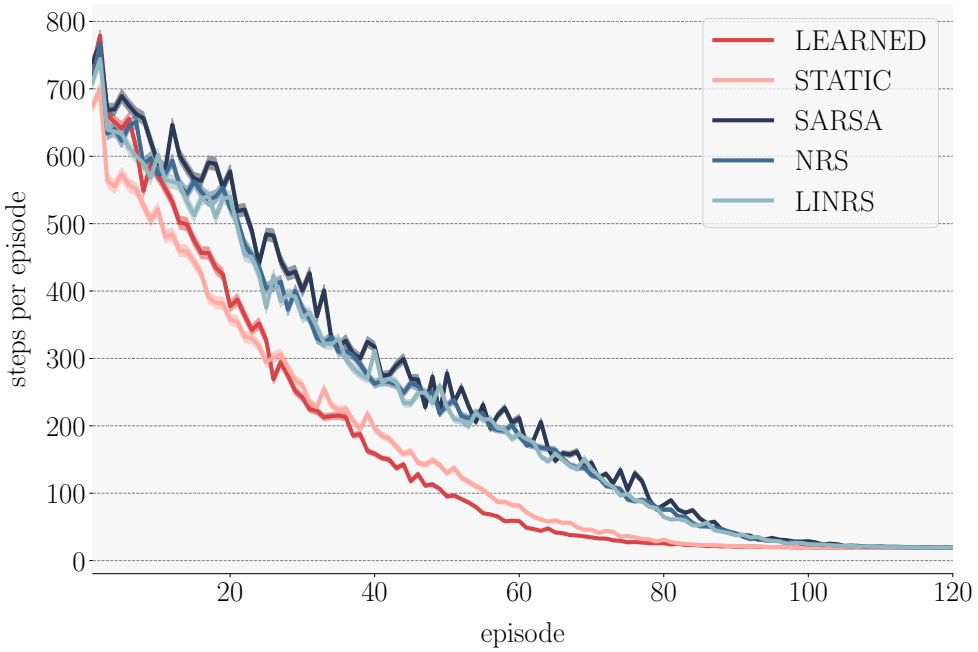


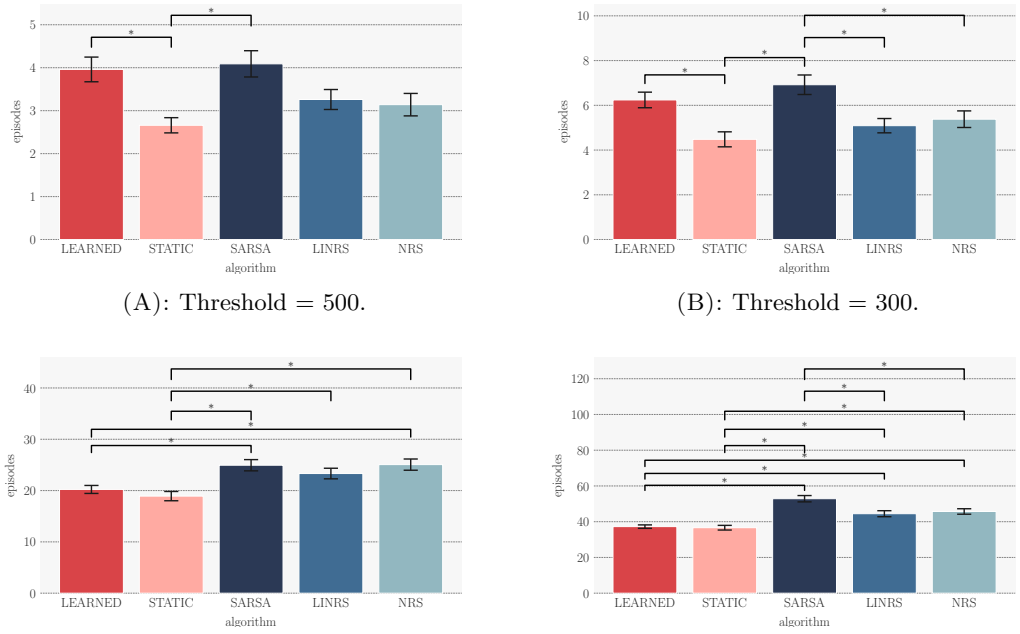
FIGURE 6.4: Learning curves compared with three methods.

the state of the four-room domain is a scalar value showing which the rectangle is. We used RANDOM including the subgoals shown in Figure 5.3(B).

6.3.4 Performance Comparison

We compare the learning performance of our methods with SARSA, NRS, and LINRS in the four-room domain in terms of Jumpstart, Time to Threshold, and Asymptotic Performance. We show the results with HUMAN in this section. Figure 6.4 shows the learning curves of LEANED, STATIC, SARSA, LINRS, and NRS.

We plotted LEARNED, STATIC, NRS, and LINRS with an average totaling 1000 learnings over all participants. SARSA was averaged in 100 learnings because it does not use the subgoals. LEARNED and STATIC seemed to be fewer steps than SARSA for almost all episodes. NRS and LINRS seemed to be almost the same steps as SARSA. The difference between subgoal-based reward shaping and the other three methods becomes large from the 10th episode. Subgoal-based reward shaping reached the bottom in the 80th episode, and the other three methods did in the 100th episode. STATIC worked more effectively than LEARNED in the 30th episode, and LEARNED overtook STATIC after the 30th episode.



(A): Threshold = 500.

(B): Threshold = 300.

(C): Threshold = 100.

(D): Threshold = 50.

FIGURE 6.5: Results of multiple comparisons among five methods in terms of Time to Threshold. Bars offer mean steps, and error bars offer standard error.

We first report the results in terms of Jumpstart. We used the number of steps in the first episode as a Jumpstart metric. We performed an ANOVA for Jumpstart, Time to Threshold, and Asymptotic Performance among LEARNED, STATIC, SARSA, NRS, and LINRS. The analysis showed no significant difference. The Holm-Bonferroni method was used for sub-effect tests. The mean (the standard deviation) Jumpstarts are 738.95 (276.83) for LEARNED, 705.49 (304.74) for STATIC, 721.49 (296.31) for SARSA, 722.76 (294.63) for NRS, and 693.66 (309.49) for LINRS.

We evaluate our methods in terms of Time to Threshold. The best performance is 19 steps, and the worst performance is 1000 steps. The learning decreases the number of steps to achieve the goal. We set the number of steps as a threshold to 500, 300, 100, and 50 to evaluate the performance during learning.

One-way ANOVA showed a significant difference in every threshold ($p < .01$), and we conducted the posthoc test at every threshold. Figure 6.5 shows the results of the multiple comparisons by Tukey's HSD method at every threshold.

Table 6.2 shows the mean episodes and the standard deviations of the compared methods for each threshold step.

We describe the results in Tables 6.2 and Figure 6.5. STATIC was the best performance 2.66 of the five methods to achieve its goal with 500 steps. SARSA took 4.09

Table 6.2: Mean and standard deviation: Mean(S.D.).

Threshold	LEARNED	STATIC	SARSA	LINRS	NRS
500	3.96 (2.86)	**2.66** (1.76)	4.09 (3.04)	3.26 (2.31)	3.14 (2.60)
300	6.24 (3.47)	**4.48** (3.34)	6.92 (4.34)	5.09 (3.18)	5.38 (3.69)
100	20.2 (7.74)	**19.9** (8.99)	24.9 (10.9)	23.3 (10.3)	25.1 (10.9)
50	37.3 (9.58)	**36.6** (13.0)	52.9 (17.6)	44.5 (16.7)	45.7 (15.2)

episodes as the worst, and LEARNED took a similar episode. STATIC was significantly fewer than LEARNED and SARSA. LINRS and NRS were significantly better performance than SARSA to reach 300 steps. The best performance in Figure 6.5(B) is 4.48 for STATIC, the worst is 6.92 for SARSA. LEARNED and STATIC took fewer episodes to reach their goals under 100 steps than SARSA and NRS. STATIC also is fewer than LINRS. The best performance is 19.9 for STATIC, and the worst is 25.1 for NRS. LEARNED and STATIC reached their goals with under 50 steps in fewer episodes than SARSA and NRS as well as LINRS. LINRS and NRS took fewer episodes than SARSA. SARSA had the worst performance 52.9. The best performance was 36.6 for STATIC. STATIC always was the fewest of the five methods. We report the results involved in Asymptotic Performance. We used the number of steps in the final episode which is the 1000th. The Asymptotic Performances are 19.0 for LINRS and NRS, 19.1 for LEARNED, SARSA, and STATIC. ANOVA revealed no significant differences among the five methods.

Our proposed methods, LEARNED and STATIC, could improve SARSA as a baseline with subgoals provided by humans in the middle of learning from the results of ANOVA and multiple comparisons in the four-room domain. Their performance in the final phase of learning was equivalent to SARSA, and they did not improve SARSA in the first phase of learning. The simple potential functions, NRS and LINRS, could improve in the middle of learning, but our methods were more effective than theirs. Though there is no significant difference, STATIC worked more efficiently than LEARNED early, and LEARNED overtook STATIC later.

6.3.5 Subgoal Quality between Human Subgoals and Random Subgoals

We compared the performance of STATIC and LEARNED with HUMAN with RANDOM. We used HUMAN as shown in Chapter 5 and RANDOM as shown in Section 6.3.3. Figure 6.6 shows the learning curves of agents using LEARNED, and Figure 6.7 shows the learning curves of the static potential. Both figures compared the case of HUMAN

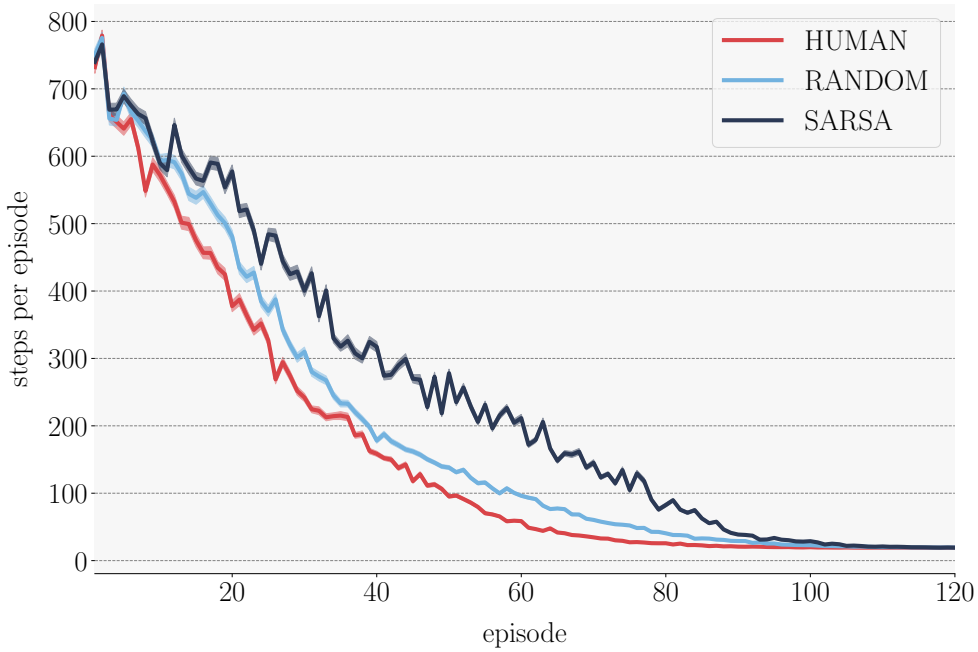


FIGURE 6.6: Learning curves of LEARNED with HUMAN and RANDOM.

TABLE 6.3: Mean and standard deviation of Jumpstart: Mean(S.D.). Bold numbers mean the minimum.

Method	HUMAN	RANDOM	SARSA
LEARNED	739.0(276.8)	803.7(262.7)	**721.5**(296.3)
STATIC	705.5(304.7)	**683.6**(310.8)	721.5(296.3)

with RANDOM and had the learning curve of SARSA to make clear the improvement in the case of RANDOM.

As shown in Figure 6.6 and 6.7, both potentials improved the learning efficiency with RANDOM compared to SARSA, but the method with HUMAN was more efficient than RANDOM. STATIC with RANDOM started deteriorating at about the 30th episode, and it became worse than SARSA after the 80th episode. We compared HUMAN with RANDOM in terms of Jumpstart. ANOVA did not reveal any significant difference between HUMAN and RANDOM both in LEARNED and STATIC. We summarized the means and standard deviations of Jumpstart in Table 6.3.

As shown in Table 6.3, the mean Jumpstarts ranged from 683.6 to 803.7, and the standard deviation was approximately one-third of its mean.

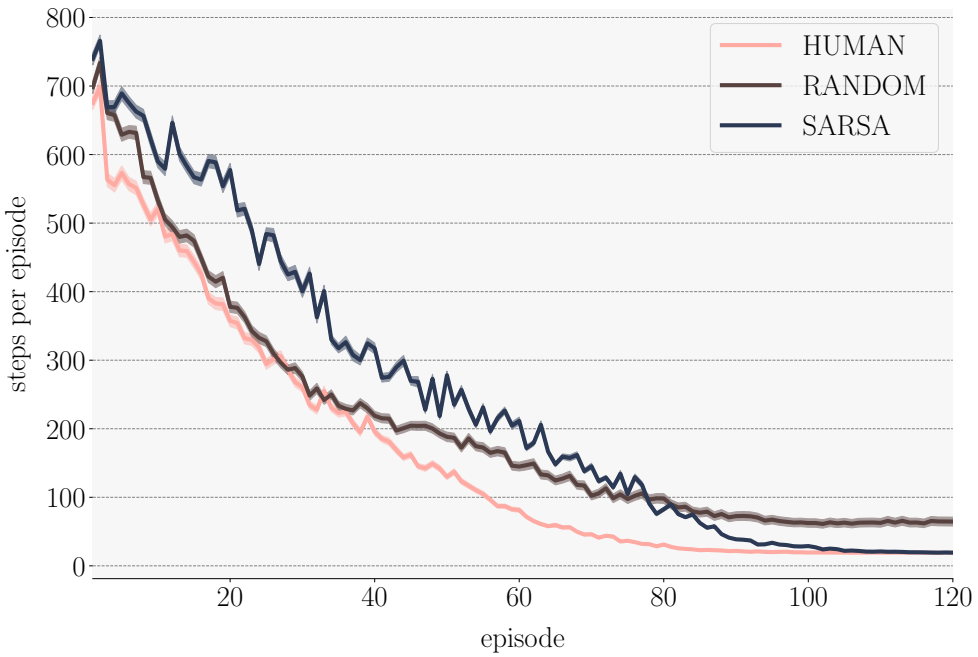


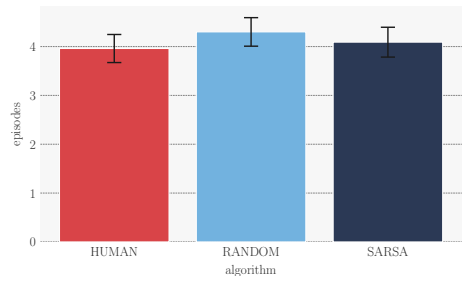
FIGURE 6.7: Learning curves of agents with STATIC with HUMAN and RANDOM.

TABLE 6.4: Mean and standard deviation of Time to Threshold in four-room domain for subgoal comparison: Mean(SD).

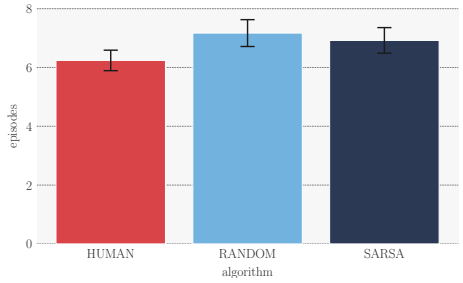
	LEARNED			STATIC	
Threshold	HUMAN	RANDOM	AC	HUMAN	RANDOM
500	**3.96 (2.86)**	4.30 (2.91)	4.09 (3.04)	**2.66 (1.76)**	3.42 (3.16)
300	**6.24 (3.47)**	7.17 (4.54)	6.92 (4.34)	**4.48 (3.34)**	6.25 (6.02)
100	**20.2 (7.74)**	25.5 (9.89)	24.9 (10.9)	**18.9 (8.99)**	22.3 (10.5)
50	**37.3 (9.58)**	45.0 (10.6)	52.9 (17.6)	**36.6 (13.0)**	48.0 (83.1)

We conducted an ANOVA and multiple comparisons for each threshold in terms of Time to Threshold. The thresholds were the same as Section 6.3.4. Figure 6.8 and 6.9 show the results of multiple comparisons for LEARNED and STATIC, respectively. $*$ shows that there was a significant difference ($p < 0.05$). ANOVA did not reveal a significant difference only in the threshold of 50. Table 6.4 shows the means and standard deviations of Time to Threshold. The bold number means the best of HUMAN, RANDOM, and AC for each potential.

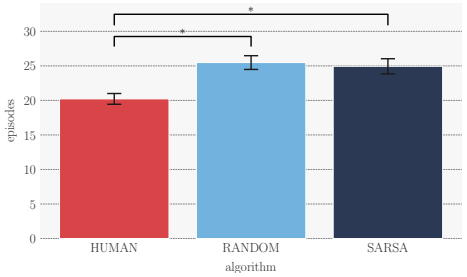
As shown in Figure 6.8, HUMAN reached the threshold at 50 and 100 in significantly fewer episodes than RANDOM. RANDOM even achieved the threshold at 50 more efficiently than SARSA. LEARNED could enable the agent to achieve 50 steps even using RANDOM, but it improved more efficiently using HUMAN in the four-room domain.



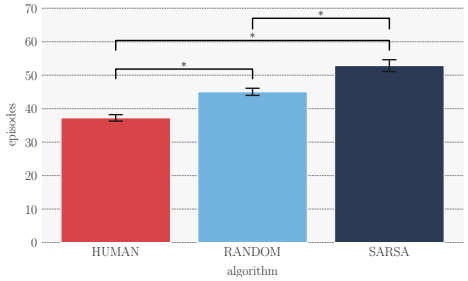
(A): Threshold = 500.



(B): Threshold = 300.

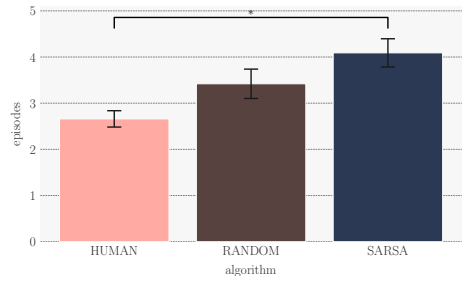


(C): Threshold = 100.

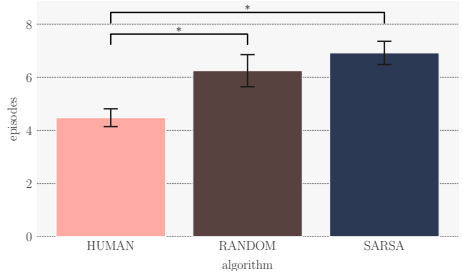


(D): Threshold = 50.

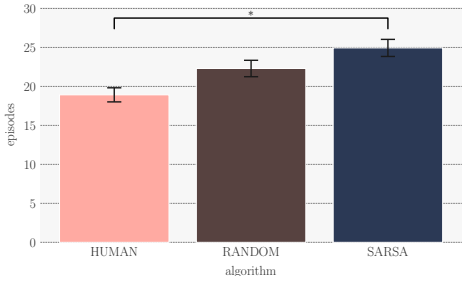
FIGURE 6.8: Results of multiple comparisons between subgoals and SARSA with LEARNED in terms of Time to Threshold. Bars offer mean steps, and error bars offer standard error.



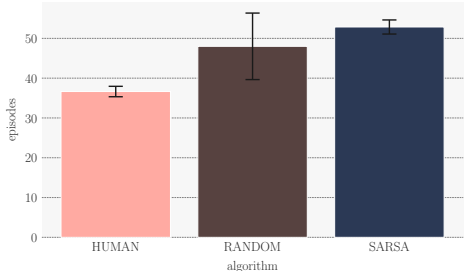
(A): Threshold = 500.



(B): Threshold = 300.

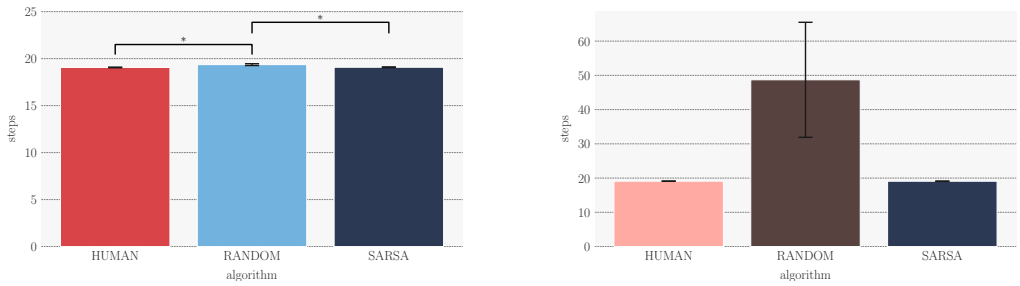


(C): Threshold = 100.



(D): Threshold = 50.

FIGURE 6.9: Results of multiple comparisons between subgoals and SARSA with STATIC in terms of Time to Threshold. Bars offer mean steps, and error bars offer standard error.



(A): LEARNED. (B): STATIC.

FIGURE 6.10: Results of multiple comparisons between subgoals and SARSA in terms of Asymptotic Performance. Bars offer mean steps, and error bars offer standard error.

STATIC with RANDOM was significantly lower than STATIC with HUMAN only in the threshold of 300 from Figure 6.9. RANDOM had a lower mean than SARSA in every threshold. HUMAN always was lower than RANDOM, and it performed significantly efficiently compared to SARSA at the thresholds of 100 and 50. STATIC also performed more efficiently using HUMAN than using RANDOM.

We finally describe the analysis in terms of Asymptotic Performance. ANOVA revealed significant differences in both LEARNED and STATIC. We summarized the results of multiple comparisons in Figure 6.10.

As shown in Figure 6.10(A), HUMAN and SARSA were significantly lower than RANDOM ($p < 0.05$) but the difference was smaller than a minimum unit of 1. Though ANOVA revealed a significant difference ($p < 0.05$), there is no significant difference among the three methods in Figure 6.10(B). RANDOM was larger than HUMAN and SARSA. RANDOM consisted of a lot of episodes with 19 steps and a few episodes with 999 steps. It means that the agent succeeded in most episodes but it sometimes failed. STATIC with RANDOM sometimes made learning unstable.

LEARNED with RANDOM even improved SARSA in the middle of learning and did not improve it in the first and last of learning. STATIC with RANDOM did not improve SARSA through learning. Though there were no significant differences, STATIC with RANDOM deteriorated SARSA.

6.4 Navigation in Pinball Domain

6.4.1 Environmental Setup

The navigation task in the pinball domain involves moving a ball to a designated target by giving it velocity. The pinball domain makes it difficult for humans to control the

ball because it needs them to control the ball with delicate actions. Delicate control is often necessary for the control domain. It is easier to give a sequence of subgoals than a trajectory in this domain.

The difference with the four-room domain is the continuous state space over the position and velocity of the ball on the x-y plane. An action space has five discrete actions, four types of force and no force. In this domain, a drag coefficient of 0.995 effectively stops ball movements after a finite number of steps when the no-force action is chosen repeatedly; collisions with obstacles are elastic. The four types of force are up, down, right, and left on a plane. An episode terminates with a reward of +10,000 when the agent reaches the goal. Interruption of any episode occurs when the episode takes more than 10000 steps. Each learning of the algorithms is repeated 100 times from scratch.

6.4.2 Algorithmic Setup

We used AC, a basic RL algorithm that optimizes a policy with parameters using the value function as the baseline RL algorithm. A parameterized policy is called an actor, and a value function is a critic. The actor and critic needed to update their parameters. We used the critic with linear function approximation over a Fourier basis [62] of order 3. We used a soft-max policy as the actor. We set the learning rates to 0.01 for the actor and critic and the discount factor to 0.99. η was set to 10,00 after a grid search on grids of 10, 100, 1,000, and 10,000. The results of the grid search were presented in section 7.2. The function $c(h)$ was almost the same as in the four-rooms domain, excluding judging whether a state is a subgoal.

6.4.3 Utilization of Subgoals

The agent used each sequence of two subgoals the participant provided in learning. The pinball domain has a four-dimensional state, but the participant could select only two-dimensional subgoals, for which two-dimension means a position. This is because the other two dimensions mean the ball's velocity, and selecting it over the web system was difficult. The subgoals have a circle-shaped range because the point in the continuous space was too small to reflect the participant's intention. The subgoals the agent used also had a circle-shaped range. The agent achieved the subgoal if it reached a position in the circle-shaped range. We set the circle's radius to 0.04, which is the same size as the goal area. We used RANDOM including subgoals shown in Figure 5.4(B) with the same setting as the participants.

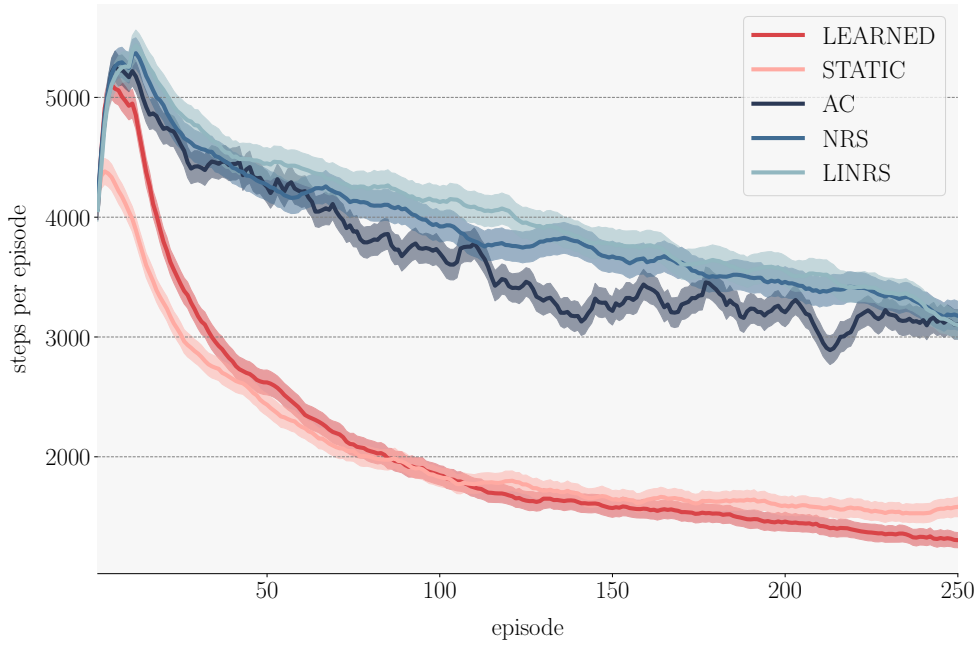


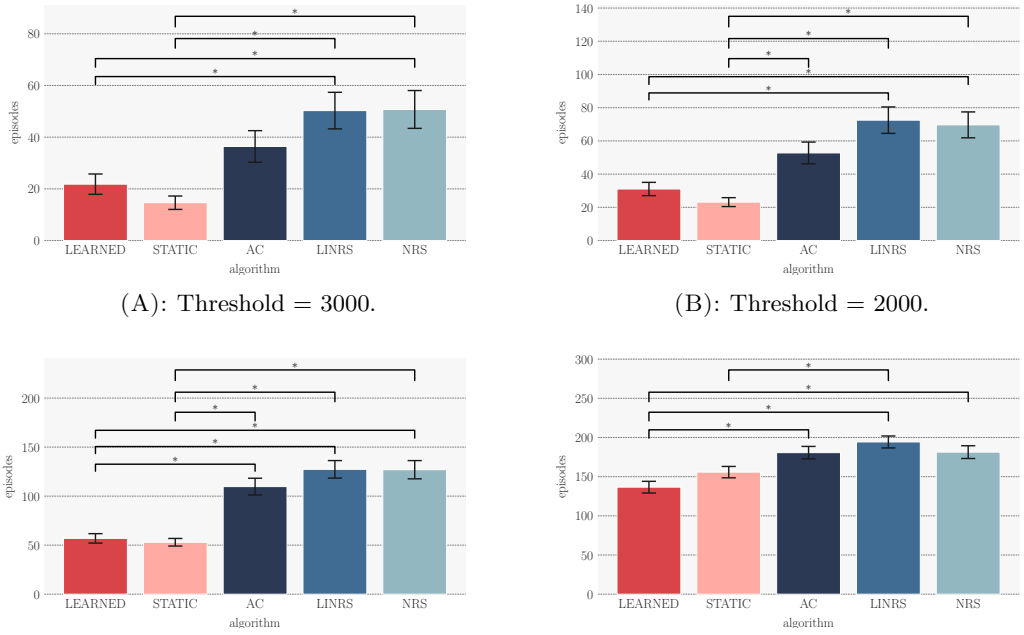
FIGURE 6.11: Learning curve in pinball domain. Lines are average shifts of mean steps. Shaded areas show standard error.

6.4.4 Performance Comparison

We compared LEARNED and STATIC with AC, NRS, and LINRS. Figure 6.11 shows the learning curves. It took an average shift of 10 episodes.

STATIC performed the best of the five methods until about the 100th episode, but LEARNED overtook it and worked efficiently in Figure 6.11. STATIC and LEARNED mostly performed better than AC, NRS, and LINRS. NRS and LINRS seemed to be less efficient than AC. NRS was a similar performance to LINRS.

We analyzed the performance in terms of Jumpstart. ANOVA did not have a significant difference among the five methods. The means and standard deviations: Mean(SD) are 3811.6(3029.6) for LEARNED, 3738.2(2999.4) for STATIC, 4246.0(2950.7) for AC, 4006.1(2859.5) for NRS, and 4608.5(3257.5) for LINRS. The best mean was STATIC and the worst was LINRS. We evaluated the learning efficiency by Time to Threshold. We used each learning result smoothed by using a simple moving average with the number of periods being 10 episodes, and we performed an ANOVA and multiple comparisons with Tukey's HSD to confirm the difference among the five methods. We set 3000, 2000, 1000, and 500 as thresholds from the learning curves shown in Figure 6.11. ANOVA revealed significant differences at every threshold. Figure 6.12 shows the results of



(A): Threshold = 3000.

(B): Threshold = 2000.

(C): Threshold = 1000.

(D): Threshold = 500.

FIGURE 6.12: Results of multiple comparisons among five methods in pinball domain in terms of Time to Threshold. Bars offer mean steps, and error bars offer standard error.

TABLE 6.5: Mean and standard deviation of Time to Threshold in pinball domain: Mean(SD).

Threshold	LEARNED	STATIC	AC	LINRS	NRS
3000	21.8 (39.2)	**14.6** (25.8)	36.4 (60.8)	50.3 (70.4)	50.7 (72.8)
2000	31.0 (39.8)	**23.2** (26.7)	52.8 (65.2)	72.5 (79.2)	69.7 (77.6)
1000	56.9 (47.8)	**53.0** (38.5)	109.7 (85.1)	127.3 (89.0)	127.0 (92.6)
500	**136.6** (74.3)	155.8 (72.2)	180.7 (79.2)	194.3 (75.9)	181.3 (80.9)

multiple comparisons with Tukey's HSD, and Table 6.5 shows the means and standard deviations.

As shown in Figure 6.12, STATIC performed significantly better than AC, NRS, and LINRS in thresholds of 3000, 2000, and 1000. STATIC was significantly better only than LINRS in a threshold of 500. LEARNED performed more efficiently than NRS and LINRS in every threshold. LEARNED reached thresholds of 1000 and 500 earlier than AC. There was no significant difference among AC, NRS, and LINRS. STATIC recorded 11.9 for a threshold of 3000 and 24.8 for a threshold of 2000 as the best. LEARNED recorded 56.9 for a threshold of 1000 and 136.6 for a threshold of 500 as the best. In contrast, LINRS worst reached thresholds of 2000, 1000, and 500 with 72.5 episodes, 127.3 episodes, and 194.3 episodes, respectively. We lastly analyzed the performance in

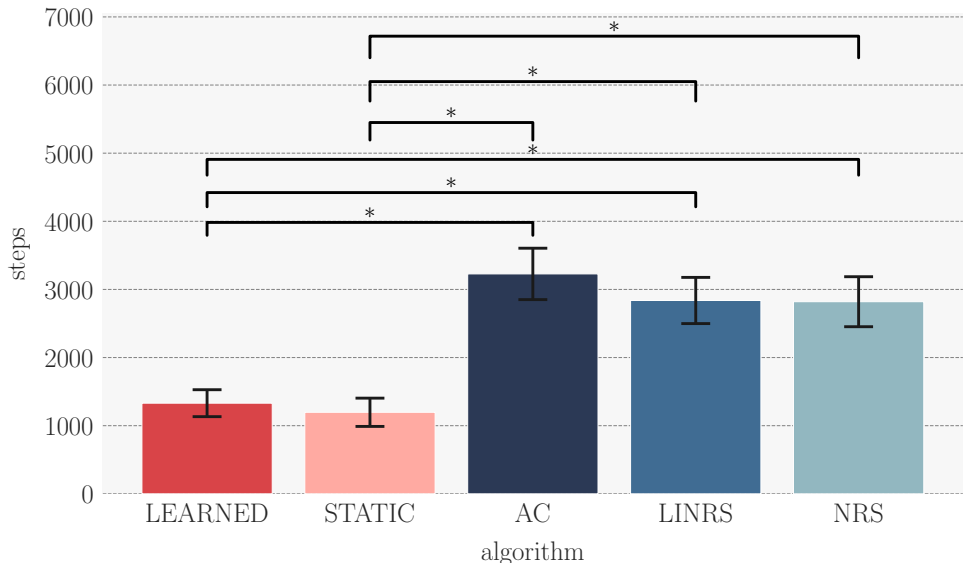


FIGURE 6.13: Results of Asymptotic Performance in pinball domain. Bars show means. Error bars show standard errors.

terms of Asymptotic Performance. ANOVA revealed a significant difference ($p < 0.01$). Figure 6.13 shows the results of multiple comparisons by bar plot.

As shown in Figure 6.13, LEARNED and STATIC were significantly lower Asymptotic Performance than the other three methods. The means and standard deviations: Mean(SD) were 1330.3(1966.9) for LEARNED, 1196.8(2065.0) for STATIC, 3227.7(3753.5) for AC, 2837.7(3373.7) for LINRS, and 2819.6(3650.9) for NRS. AC was the highest of the five methods.

STATIC and LEARNED could improve AC from the middle, but not in the earlier learning from ANOVA and multiple comparisons. The simple potential functions, NRS and LINRS, could not improve AC through learning.

6.4.5 Subgoal Quality between Human Subgoals and Random Subgoals

This section compared HUMAN with RANDOM for STATIC and LEARNED in terms of learning efficiency. Figure 6.14 and 6.15 show the learning curves of LEARNED and STATIC, respectively. We also plotted the learning curve of AC in both figures. HUMAN of Figure 6.14 is the same as LEARNED of Figure 6.11, and HUMAN of Figure 6.15 is the same as STATIC of Figure 6.11.

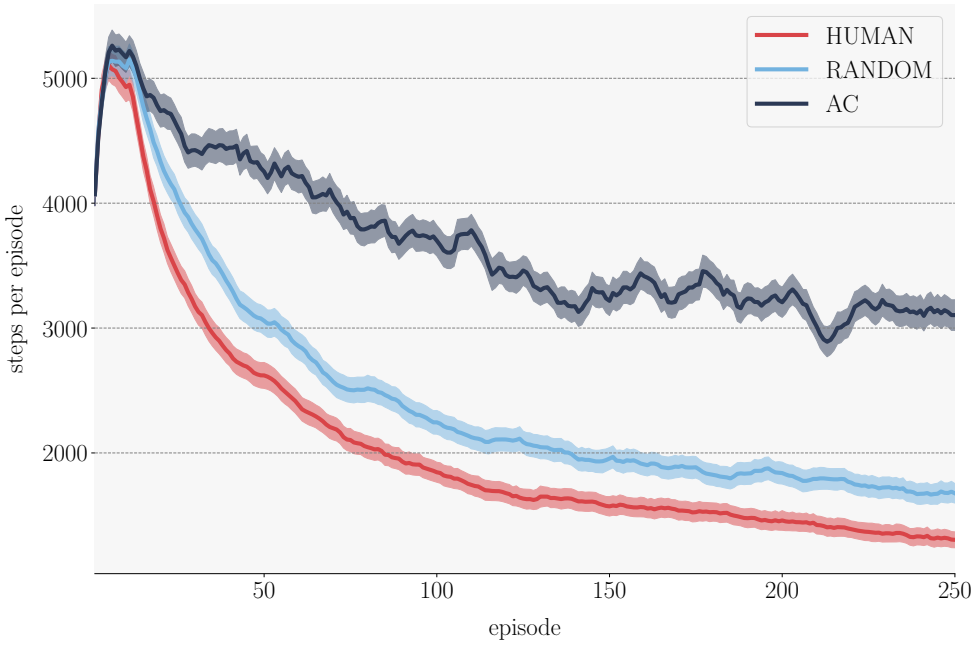


FIGURE 6.14: Learning curves of LEARNED in pinball domain.

TABLE 6.6: Mean and standard deviation of Jumpstart: Mean(S.D.). Bold numbers mean the minimum.

Method	HUMAN	RANDOM	SARSA
LEARNED	**3811.6** (3029.6)	**3811.6** (3029.6)	4246.0 (2950.8)
STATIC	**3738.2** (2999.4)	4304.8 (3172.3)	4246.0 (2950.8)

RANDOM was more efficient than AC, it took more steps than LEARNED in every episode in Figure 6.14. RANDOM also performed more efficiently than AC in Figure 6.15. RANDOM was worse than LEARNED through learning.

We analyzed the performances of HUMAN and RANDOM in terms of Jumpstart. ANOVA did not have significant differences among HUMAN, RANDOM, and AC in both LEARNED and RANDOM. We summarized the means and standard deviations of Jumpstart in Table 6.6

As shown in Table 6.6, HUMAN and RANDOM were the lowest means at 3811.6 in a method of LEARNED. HUMAN was the lowest mean at 3738.2 in a method of STATIC. SARSA was the same mean in both methods because it did not use the subgoals. We analyzed the performances of HUMAN and RANDOM in every threshold of Time to Threshold. We used the same thresholds in Section 6.4.4. ANOVA revealed significant

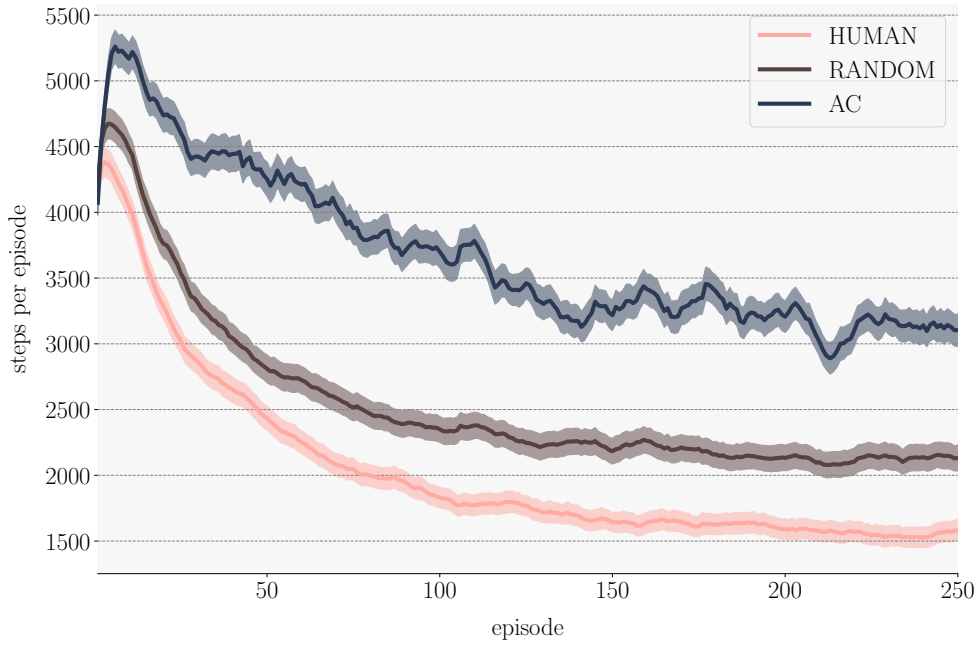


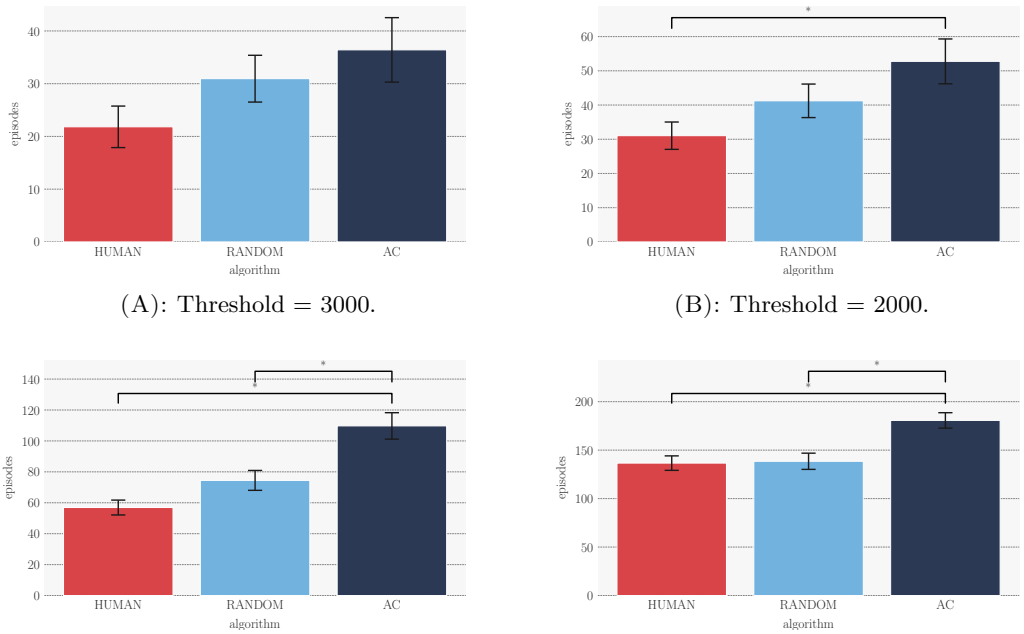
FIGURE 6.15: Learning curves of STATIC in pinball domain.

TABLE 6.7: Mean and standard deviation of Time to Threshold in pinball domain for subgoal comparison: Mean(SD).

	LEARNED			STATIC	
Threshold	HUMAN	RANDOM	AC	HUMAN	RANDOM
3000	**21.8 (39.2)**	31.0 (44.1)	36.4 (60.8)	**14.6 (25.8)**	23.2 (42.6)
2000	**31.0 (39.8)**	41.2(48.7)	52.8 (65.2)	**23.2 (26.7)**	39.0 (50.8)
1000	**56.9 (47.8)**	74.5 (63.9)	110 (85.1)	**53.0 (38.5)**	70.3 (38.5)
500	**137 (74.3)**	139(83.2)	181 (79.2)	155 (72.2)	**153.0 (82.0)**

differences in thresholds except for 3000 in LEARNED. STATIC had significant differences in every threshold. Figure 6.16 and (B) show bar plots with the results of multiple comparisons in LEARNED and STATIC, respectively. Table 6.7 summarizes the means and standard deviations.

As shown in Figure 6.16, HUMAN was significantly lower than AC in thresholds of 2000, 1000, and 500. RANDOM was significantly lower than AC in thresholds of 1000 and 500. RANDOM reached the thresholds earlier than AC in all thresholds but it included non-significant differences. HUMAN was the best Time to Threshold of the three methods in all thresholds. From Figure 6.17, HUMAN and RANDOM were significantly lower than AC in a threshold of 1000, and HUMAN was only significantly lower than AC



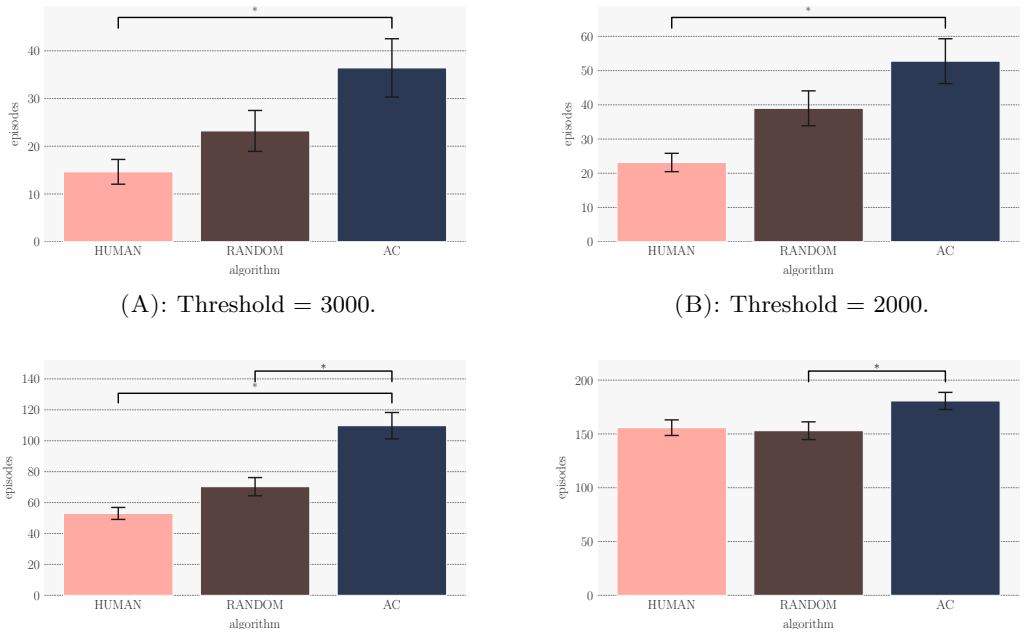
(A): Threshold = 3000.

(B): Threshold = 2000.

(C): Threshold = 1000.

(D): Threshold = 500.

FIGURE 6.16: Results of multiple comparisons between LEARNED with subgoals and AC in pinball domain in terms of Time to Threshold. Bars offer mean steps, and error bars offer standard error.



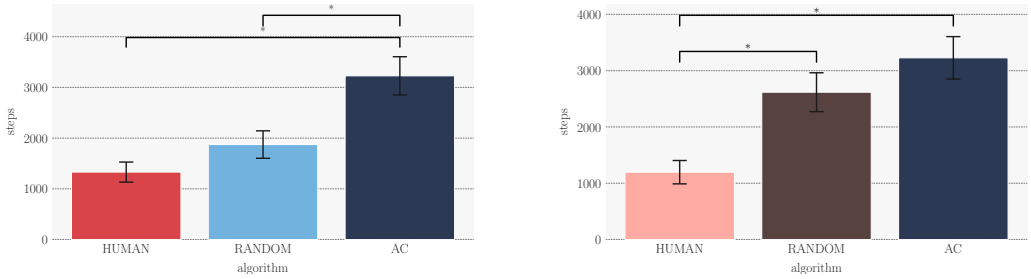
(A): Threshold = 3000.

(B): Threshold = 2000.

(C): Threshold = 1000.

(D): Threshold = 500.

FIGURE 6.17: Results of multiple comparisons between subgoals and AC with STATIC in pinball domain in terms of Time to Threshold. Bars offer mean steps, and error bars offer standard error.



(A): LEARNED (B): STATIC

FIGURE 6.18: Results of multiple comparisons between subgoals and AC in pinball domain in terms of Asymptotic Performance. Bars offer mean steps, and error bars offer standard error.

in thresholds of 3000, 2000, and 1000. HUMAN was the best Time to Threshold of the three methods in thresholds except for 500. Table 6.7 shows a difference between HUMAN and RANDOM increased in thresholds from 3000 to 1000, and then it decreased at a threshold of 500 in both LEARNED and STATIC. The maximum differences were recorded at a threshold of 500 by nearly 20 episodes($74.5 - 56.9 = 17.6$ for LEARNED, $70.3 - 53.0 = 17.3$ for STATIC).

We compared the subgoals in terms of Asymptotic Performance. There was a significant difference only in LEARNED by ANOVA ($p < 0.01$). Figure 6.18 shows bar plots with the results of multiple comparisons in LEARNED and STATIC.

The means and standard deviations of RANDOM: Mean(SD) were 1873(2694) for LEARNED and 2617(3439) for STATIC. We reported the other means and standard deviations in Section 6.4.4. HUMAN and RANDOM were significantly lower than AC in Figure 6.18. The lowest mean was HUMAN, and the highest was AC in Figure 6.18(A). RANDOM was the lowest, and AC was the highest in Figure 6.18(B).

We summarized the comparison between HUMAN and RANDOM in the pinball domain. LEARNED with RANDOM and STATIC with RANDOM could significantly improve AC in the middle of learning and could not work efficiently in the earlier learning. LEARNED with RANDOM could learn more effective policy than AC. Though there were no significant differences, HUMAN worked more effectively than RANDOM.

6.5 Pick and Place Task with Fetch Robot

6.5.1 Environmental Setup

We used a pick-and-place domain, where the robotic arm learns to grasp a box and move it to a target position[23], with the 7-DoF Fetch robotics arm of OpenAI Gym[63]. We converted the original task into a single-goal reinforcement learning framework because potential-based reward shaping is not compatible with multiple goals [7]. The dimension of observation is bigger than the previous navigation tasks, and the action is continuous. An observation is 25-dimensional and includes the Cartesian position of the gripper and the object and the object's position relative to the gripper. The reward function generates a reward of +1 when the box is in the target position and no reward at every step. The task ends at 50 steps regardless of success or failure. We separated the policy in a training phase from in a test phase. The policy made a ϵ-greedy selection and added noise to a given noise scale. The agent selected an action without exploration and noise in the test phase. The agents generated an episode of test per four episodes of training. The agent updated every training episode after the first 256 steps. The agent repeated an iteration of the four training and the test episode 15000 times. The agent learned five times on different random seeds. We parallelized learning with ten cores by MPI [64]. We synchronized the initial parameters and the gradients between the cores. The metrics also were averaged over the cores.

6.5.2 Algorithmic Setup

The learning in 200 epochs took several hours. We implemented based on an OpenAI Baselines[65] implementation for DDPG [42] as the baseline. We also implemented Random Network Distillation (RND) [66] to let the agent explore effectively in the environment with a scant reward. We developed them with PyTorch [67]. We used the same network architecture for the policy and the q-value. The network had three hidden layers, and all the hidden layers had 256 nodes. The output function of the policy network was a hyperbolic tangent, and the q-value function used a linear function. Both outputs were one dimension. The network of RND had two hidden layers, each with 256 nodes. All the networks were learned with Adam optimizer [68] and ReLU activation function [69]. The policy was updated with an L2-norm penalty. The q-value network synchronized with the target q-network every fifty updates with the Polyak-Ruppert averaging [70]. We built a table-formed value function over abstract states for LEARNED and initialized the values as zero.

Table 6.8: Summary of hyperparameter-tuning

hyperparameter	best result	search space
γ	0.9	$\{0.9, 0.99, 0.999, 0.9999, 0.95, 0.995, 0.98\}$
batch size	512	$\{128, 256, 512, 1024\}$
ϵ	0.3	$\{0.1, 0.2, 0.3, 0.4, 0.5\}$
learning rate of policy	0.0001	$\{0.01, 0.001, 0.005, 0.0001, 0.00001\}$
learning rate of q-value	0.0001	$\{0.01, 0.001, 0.005, 0.0001, 0.00001\}$
replay size	100000	$\{10000, 100000, 1000000\}$
polyak	0.92	$\{0.95, 0.98, 0.92\}$
L2	0.8	$\{0.5, 0.8, 1.0, 1.2, 1.5\}$
noise-scale	0.2	$\{0.2, 0.3, 0.1, 0.4, 0.01\}$
feature size	128	$\{32, 64, 128, 256, 512\}$
learning rate of rnd	0.01	$\{0.01, 0.001, 0.005, 0.0001, 0.00001\}$
learning rate of LEARNED	0.005	$\{0.01, 0.001, 0.005, 0.0001, 0.00001\}$
η	1	$\{0.001, 0.01, 0.1, 1, 10, 100\}$

Table 6.9: Programmatic conditions under which agent achieves subgoals. all function outputs true if all dimensions are true and outputs false otherwise.

Subgoal in writing	Programmatic condition
The robot arm reaches the location available to grasp the object.	$\text{all}\left(-\vec{\Delta}_{r,o} \leq \vec{d}_{r,o} \leq \vec{\Delta}_{r,o}\right)$
The robot arm grasps the object.	$\text{all}\left(-\vec{\Delta}_{r,o} \leq \vec{d}_{r,o} \leq \vec{\Delta}_{r,p}\right)$ && $\text{all}\left(-\vec{\Delta}_{g} \leq \vec{d}_{g} \leq \vec{\Delta}_{g}\right)$

We tuned hyperparameters with a Tree-structured Parzen Estimator sampler [71] implemented as default in Optuna [72] for 200 trials. We summarize the hyperparameters, each search space, and the results of the hyperparameter-tuning in Table 6.8.

6.5.3 Utilization of Subgoals

The agent used each sequence of two subgoals the participant provided in learning. We collected the subgoals that a participant depicted in writing for the pick-and-place domain. We transformed the writing into programmatic conditions by hand. Though some methods studied in Natural Language Processing automatically extract subgoals from sentences [73], we did not use them in order to focus on utilizing the extracted subgoals. The state included the three-dimensional distance vector between the robot arm and the object ($\vec{d}_{r,o}$) and the gap between grippers ($\vec{d}_g$). We express the programmatic conditions under which the agent achieves the subgoals as shown in Table 6.9.

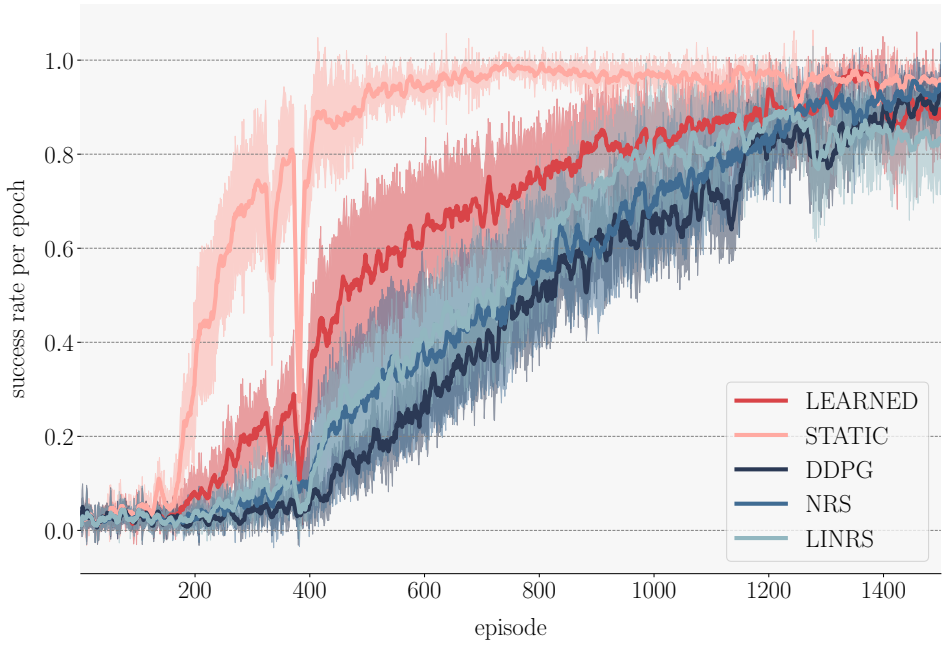


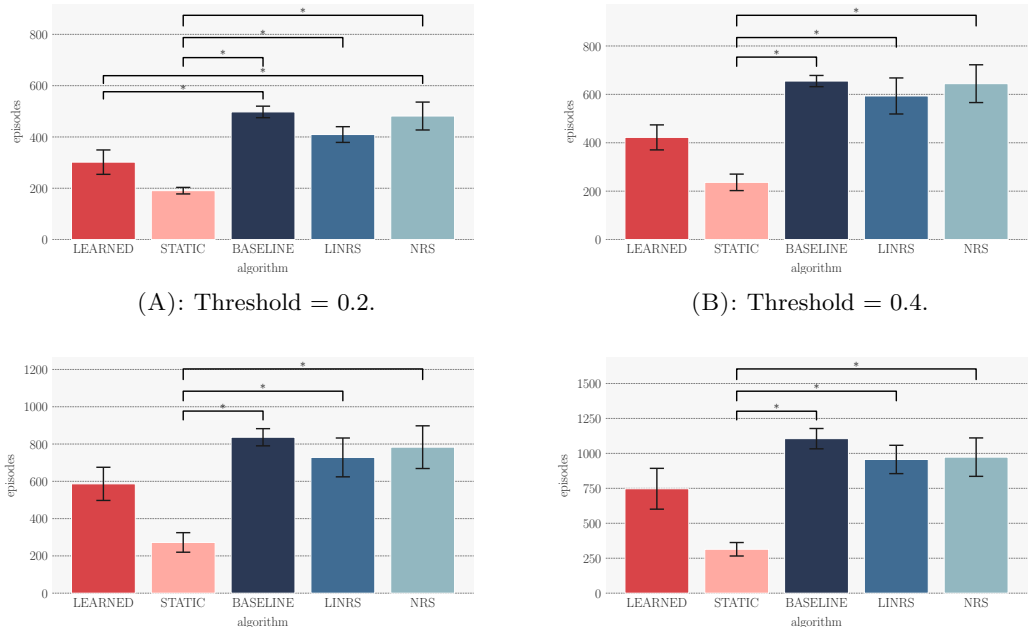
FIGURE 6.19: Learning curves in pick-and-place task.

The pick-and-place domain has a continuous state space, so we set the range for the conditions because the point in the continuous space is too small for the agent to reach, similar to the pinball domain. We denote the three-dimensional range vector for $\vec{d}_{r,o}$ as $\vec{\Delta}_{r,o}$ and two-dimensional range vector for $\vec{d}_g$ as $\vec{\Delta}_g$, and we set $\vec{\Delta}_{r,o}$ for $\{0.07, 0.07, 0.07\}$ and $\vec{\Delta}_g$ for $\{0.005, 0.005\}$. We sampled all subgoals from NumPy's uniform distribution [58]. We also decided on the order randomly. The subgoals of RANDOM were a set of vectors ranging from -5 to 5 in the state space. We set $\Delta_{r,o}$ and Δ_g the same as HUMAN.

6.5.4 Performance Comparison

We compared the learning performance of STATIC, LEARNED with DDPG, NRS, and LINRS in the pick-and-place domain in terms of Jumpstart, Time to Threshold, and Asymptotic Performance. Figure 6.19 shows the learning curves of LEARNED, STATIC, DDPG, LINRS, and NRS. The learning curves were taken moving averages over 10 points. We plotted the learning curves averaged across five learnings, and the shaded areas represent standard errors.

As shown in Figure 6.19, STATIC seems to be the most effective of all the methods, especially after nearly the 200th episode. LEARNED seems to learn more effectively than DDPG, NRS, and LINRS after nearly the 200th episode. The performances both



(A): Threshold = 0.2.

(B): Threshold = 0.4.

(C): Threshold = 0.6.

(D): Threshold = 0.8.

FIGURE 6.20: Results of multiple comparisons among five methods in pick-and-place domain in terms of Time to Threshold. Bars offer mean steps, and error bars offer standard error.

TABLE 6.10: Mean and standard deviation in pick-and-place: Mean(S.D.).

Threshold	LEARNED	STATIC	DDPG	LINRS	NRS
0.2	301 (95.1)	**190** (25.5)	497 (44.9)	409 (61.3)	481 (108)
0.4	422 (103)	**236** (68.1)	655 (46.3)	593 (149)	644 (156)
0.6	586 (178)	**272** (105)	836 (92.1)	728 (208)	783 (229)
0.8	747 (291)	**314** (96.4)	1105 (145)	956 (203)	973 (275)

of NRS and LINRS were similar to DDPG. LINRS was a little better than DDPG from the 800th to the 1200th episode. We analyzed the results with the Jumpstart metric. ANOVA revealed no significant difference among the three levels. The means(the standard deviation) of Jumpstart for all methods were 0.02(0.04). We did not find differences in the initial phase of learning. We analyzed the performances of the five methods in terms of Time to Threshold. We set thresholds as 0.2, 0.4, 0.6, and 0.8 from the learning curves in Figure 6.19. ANOVA revealed significant differences in every threshold ($p < 0.01$). Figure 6.20 shows the results of multiple comparisons by Tukey's HSD, and Table 6.10 shows the mean episodes and the standard deviations of the compared methods for each threshold step.

As shown in Figure 6.20, STATIC performed better than the other methods except for LEARNED in all the thresholds. LEARNED reached a threshold of 0.2 in fewer episodes

than DDPG and NRS. LINRS and NRS did not have significant differences from DDPG in all thresholds but had higher performance than DDPG, and LINRS reached fewer episodes than NRS. LEARNED decreased the number of episodes compared to DDPG by 196 at a threshold of 0.2, and STATIC improved DDPG by 791 at a threshold of 0.8.

We compared the performances of the five methods in terms of Asymptotic Performance. ANOVA did not find a significant difference. The means and standard deviations expressed by Mean(SD) were 0.96(0.08) for LEARNED, 1.00(0.00) for STATIC, 0.92(0.12) for DDPG, 0.90(0.20) for NRS, and 0.98(0.04) for LINRS. DDPG was the highest mean of 1.00 and NRS was the lowest mean of 0.90. All the means were similar, and the learned agent acted almost successfully.

STATIC and LEARNED did not improve the performance in the first and last phases and could improve the performance compared to DDPG in the middle of learning in the pick-and-place domain.

6.5.5 Subgoal Quality between Human Subgoals and Random Subgoals

This section shows the difference in quality between HUMAN and RANDOM in the pick-and-place domain. This section also evaluated the performance by Jumpstart, Time to Threshold, and Asymptotic Performance. Figure 6.21 shows the learning curves of LEARNED with subgoals acquired in two different ways.

As shown in Figure 6.21, The performances of HUMAN and RANDOM were similar. Both could improve DDPG. Figure 6.22 shows the learning curves of STATIC with subgoals acquired in two different ways.

As shown in Figure 6.22, RANDOM was not efficient, and it had a similar performance to DDPG in almost episodes. RANDOM was more efficient than DDPG until the 600th episode, and then it was close to DDPG. The performance of RANDOM seems to drastically fall more times than HUMAN and DDPG.

We analyzed the performances of the subgoals in terms of Jumpstart in LEARNED and STATIC. ANOVA did not find significant differences in LEARNED and STATIC. The means and standard deviations of Jumpstart with RANDOM expressed by Mean(SD) were 0.02(0.04) for LEARNED and 0.04(0.08) for STATIC. We reported the other means and standard deviations in Section 6.5.4. We compared HUMAN with RANDOM for LEARNED and STATIC in terms of Time to Threshold. We used the same thresholds as Section 6.5.4. ANOVA of LEARNED found significant differences in thresholds of

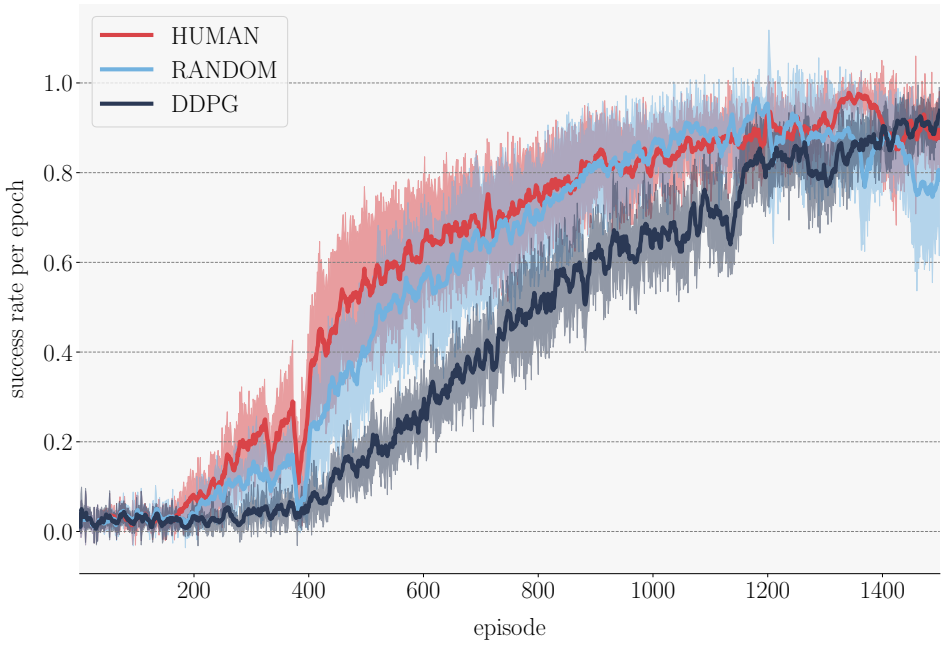


FIGURE 6.21: Learning curves with LEARNED in pick-and-place task.

0.2 ($p < 0.05$) and 0.4 ($p < 0.01$). Figure 6.23 shows the barplot of Time to Threshold to compare RANDOM for LEARNED with HUMAN for LEARNED and DDPG.

As shown in Figure 6.23, only HUMAN performed significantly better than DDPG at thresholds of 0.2 and 0.4. Though RANDOM did not have a significant difference, RANDOM improved DDPG but reached every threshold in more episodes than HUMAN. ANOVA of STATIC revealed significant differences in every threshold. Figure 6.24 shows the results of multiple comparisons by Tukey's HSD method.

As shown in Figure 6.24, HUMAN reached thresholds earlier than RANDOM and DDPG in every threshold. RANDOM was better than DDPG in thresholds of 0.2 and 0.8. RANDOM even could lead the agent to reach success rates of 0.2 and 0.8. The standard errors of RANDOM were larger than HUMAN and DDPG in every threshold. Table 6.11 shows the means and standard deviations of Time to Threshold in LEARNED and STATIC.

The maximums of difference between HUMAN and RANDOM were $59(= 806 - 747)$ for LEARNED and $586(= 900 - 314)$ for STATIC at a threshold of 0.8 from Table 6.11. The difference between HUMAN and RANDOM in LEARNED was fewer than in STATIC. RANDOM even could improve AC by 299 for LEARNED and 205 for STATIC at a threshold of 0.8.

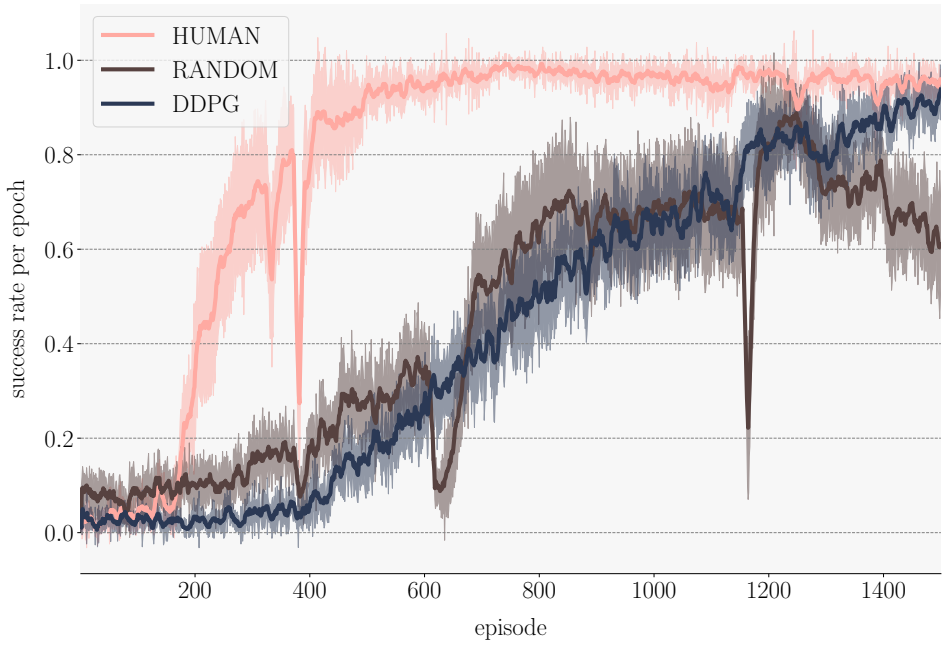


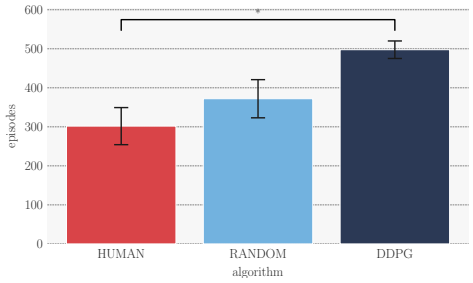
FIGURE 6.22: Learning curves with STATIC in pick-and-place task.

TABLE 6.11: Mean and standard deviation of Time to Threshold in pinball domain for subgoal comparison: Mean(SD).

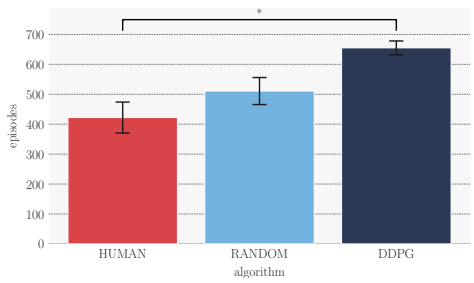
	LEARNED			STATIC	
Threshold	HUMAN	RANDOM	AC	HUMAN	RANDOM
0.2	**301 (95.1)**	372 (97.8)	498 (44.9)	**191 (25.5)**	307 (83.5)
0.4	**442 (103)**	510 (90.4)	655 (46.3)	**236 (68.1)**	587 (134)
0.6	**586 (178)**	643 (168)	836 (92.1)	**272 (105)**	800 (178)
0.8	**747 (292)**	806 (145)	1105 (145)	**314 (96.4)**	900 (204)

We analyzed the performances of subgoals in terms of Asymptotic Performance. ANOVA did not find significant differences in both LEARNED and STATIC. The means and standard deviations of Asymptotic Performance for RANDOM expressed by Mean(SD) were 0.86(0.10) for LEARNED and 0.72(0.28) for STATIC. The other means and standard deviations were reported in Section 6.5.4. Though there was no significant difference, STATIC with RANDOM and LEARNED with RANDOM arose lower Asymptotic Performance than those with HUMAN and DDPG.

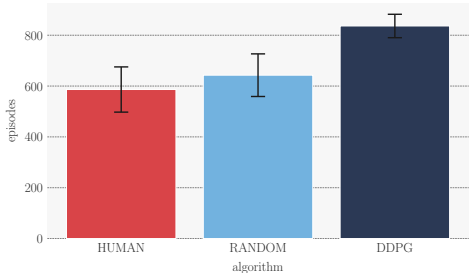
There were no significant differences between RANDOM and HUMAN and between RANDOM and DDPG through learning. RANDOM could improve the performance compared to DDPG when an agent used LEARNED. In contrast, STATIC with RANDOM could not improve DDPG.



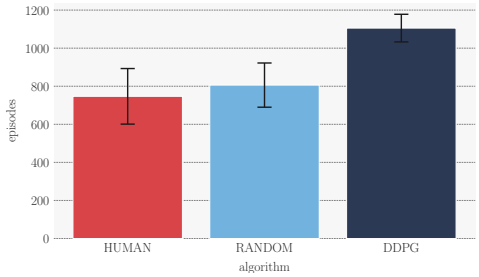
(A): Threshold = 0.2.



(B): Threshold = 0.4.

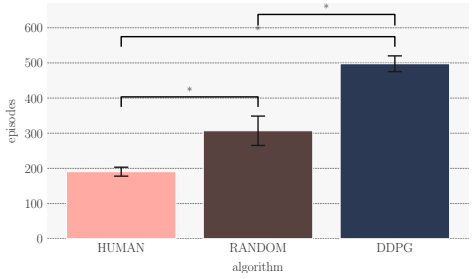


(C): Threshold = 0.6.

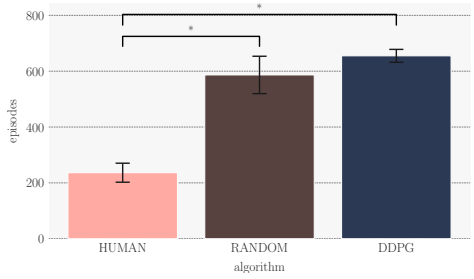


(D): Threshold = 0.8.

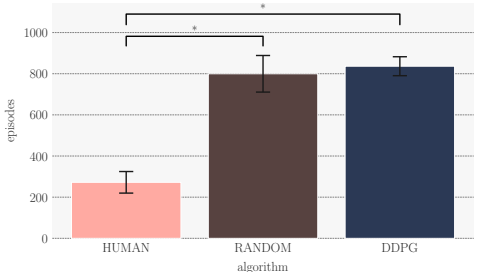
FIGURE 6.23: Results of multiple comparisons between subgoals and DDPG with LEARNED in pick-and-place domain in terms of Time to Threshold. Bars offer mean steps, and error bars offer standard error.



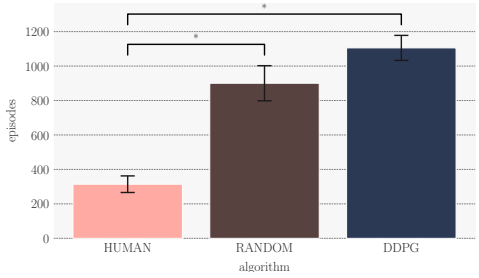
(A): Threshold = 0.2.



(B): Threshold = 0.4.



(C): Threshold = 0.6.



(D): Threshold = 0.8.

FIGURE 6.24: Results of multiple comparisons between subgoals and DDPG with STATIC in pick-and-place domain in terms of Time to Threshold. Bars offer mean steps, and error bars offer standard error.

6.6 Summary

This section summarizes the experimental results in four-room, pinball, and pick-and-place domains. In common with all three domains, STATIC and LEARNED with HUMAN could improve the baseline algorithm in terms of learning efficiency in the middle of learning. In contrast, the simple potentials could not improve the baseline. This means that intuitive methods do not work well. LEARNED with RANDOM even could improve the baseline algorithm in all three domains. STATIC sometimes could not improve the baseline algorithm. It means that LEARNED is robust to subgoal quality. This result emphasizes the strength of LEARNED compared to STATIC. STATIC worked well drastically in the pick-and-place domain, unlike the four-room and pinball domains. We analyzed it in Chapter 7. STATIC worked most efficiently and LEARNED overtook it in the four-room and pinball domains. We interpret it as that LEARNED learns its potential and it affects the performance after the learning is enough. STATIC does not learn, so it can affect performance early. HUMAN always has a better effect on subgoal-based reward shaping than RANDOM in all three domains.

Chapter 7

General Discussion

This chapter presents the discussions in terms of subgoal-based reward shaping. Section 7.1 analyzes the dependency of subgoals, and section 7.2 discusses the hyperparameters for the potentials in subgoal-based reward shaping. In section 7.3, we investigate the performance of subgoal-based reward shaping under several types of reward functions. Section 7.4 shows the limitations.

7.1 Dependency of subgoals

7.1.1 Analyzing Performances between Huamn and Random Subgoals

There was a small difference between HUMAN and RANDOM in the four-room domain from Figure 6.6 and 6.7and the pinball domain from Figure 6.14 and 6.15. The difference between HUMAN and RANDOM was large in the pick-and-place domain. Approximately 65% of states generated randomly were in an optimal trajectory for the four-room domain. The pinball domain seemed to have approximately 20% of states generated randomly in the optimal trajectory. There was no state generated randomly in an optimal trajectory in the pick-and-place domain. The random subgoal sequences were better in the four-room and pinball domains than in the pick-and-place domain. This is because these two domains might have had more states in the optimal trajectory than the pick-and-place domain. We think that the small difference between HUMAN and RANDOM was caused by the domains with states in the optimal trajectory. The

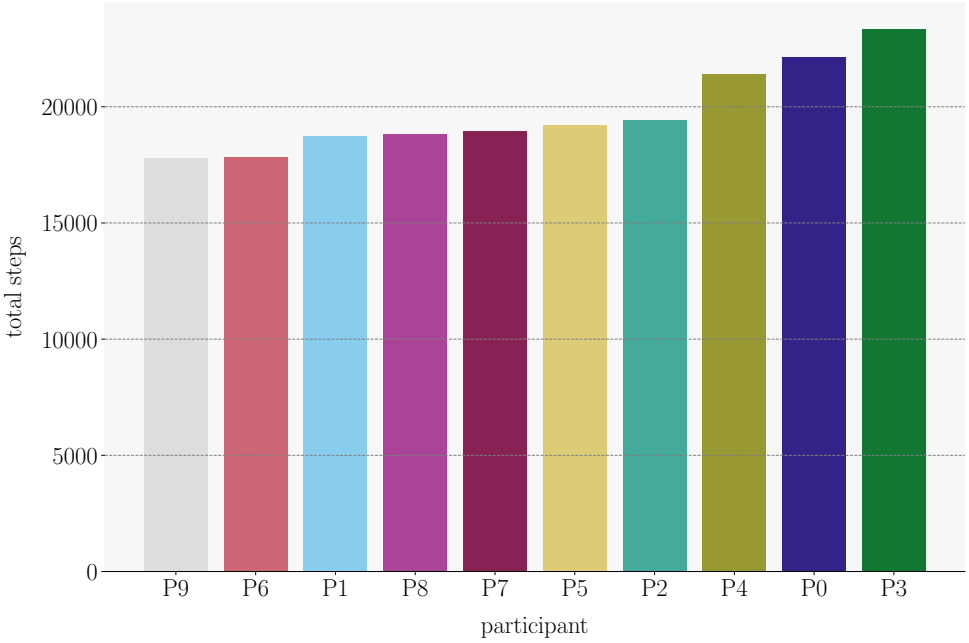
pick-and-place domain has a large state space, so the random subgoals are not likely to be placed on the optimal trajectories. The larger the state space becomes, the larger the difference between HUMAN and RANDOM may become.

Potential-based reward shaping keeps a policy invariant from the transformation of a reward function. From the experimental results in the pinball domain, the asymptotic performances of HSRS were statistically significantly lower than RSRS. There was no significant difference in the four-room domain. As shown in Figure 6.7, the performance was asymptotic at the 121st of 1000 episodes. The learning was asymptotic as the 200th episode for the pinball domain in Figure 6.15. Since our proposed method is based on PBRS, RSRS converges to the same performance as HSRS if learning continues.

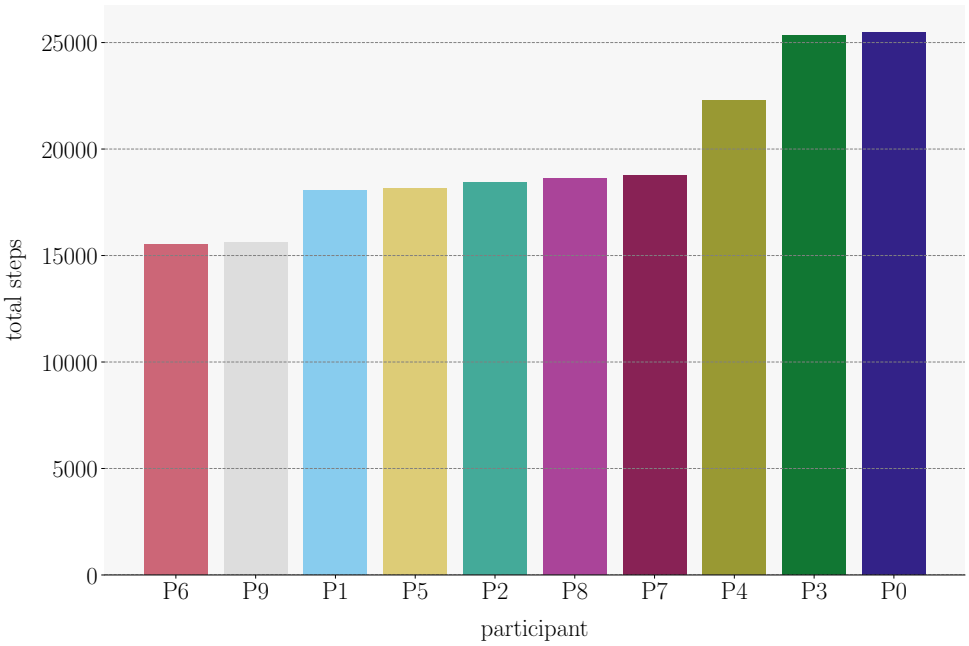
7.1.2 Subgoal Quality among Participants

This section reports the subgoal quality for subgoal-based reward shaping among participants. An agent with a subgoal-based reward shaping with both potentials learned using each participant's subgoal sequence. We evaluated LEARNED with the averaged performance metrics over every subgoal sequence acquired from participants, as mentioned in Chapter 6. Ten participants gave a subgoal sequence in the four-room domain and the pinball domain, and five participants gave them in the pick-and-place domain. We did not analyze the subgoal quality of the pick-and-place domain because there was a single pattern of a subgoal sequence. We compared the learning efficiency of LEARNED and STATIC among participants in terms of total performance accumulated during learning [59]. Figure 7.1 shows bar plots of LEARNED and STATIC among the participants in the four-room and pinball domain. We used the number of steps taken until the goal position as the performance metric and summed the number of steps until the 120th episode. Many learning curves were asymptotic in the 120th episode. A lower value was better which means that the learning was efficient. A better subgoal given by a participant was plotted to the left in the figure by sorting the performance. "P" stands for a participant, and P1 means the 1st participant.

The participants who acquired the best subgoal sequence were P9 for LEARNED and P6 for STATIC, and the participants who acquired the worst subgoal sequence were P3 for LEARNED and P0 for STATIC, as shown in Figure 7.1. Though participants with the best and worst subgoal sequences were different, the ranking of LEARNED was similar to STATIC. The differences between the best and worst subgoal sequences were 5538 steps for LEARNED and 9950 steps for STATIC, and STATIC had a larger difference between the best and worst subgoal sequences than LEARNED. The best

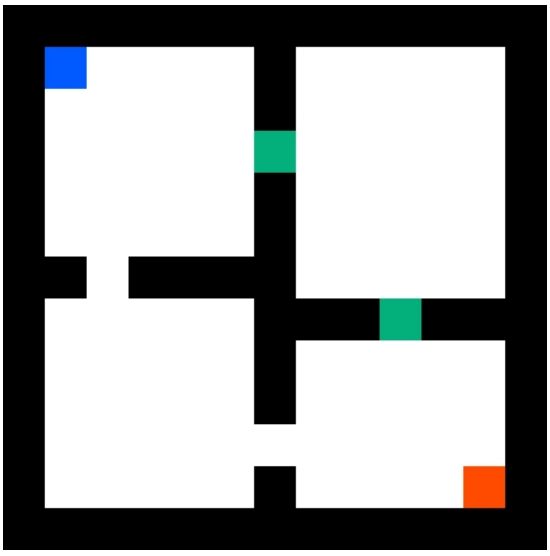


(A): LEARNED.

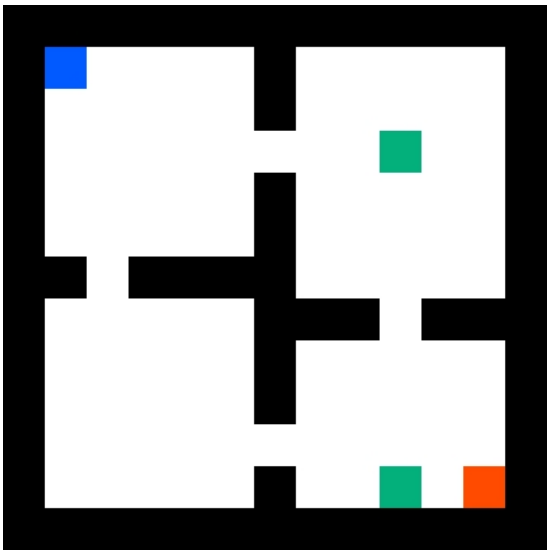


(B): STATIC.

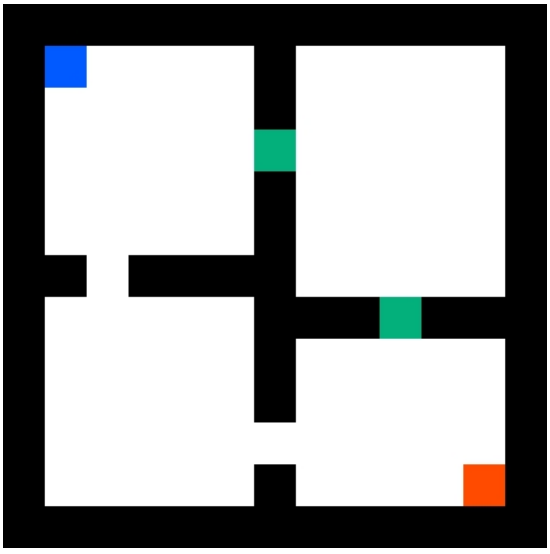
FIGURE 7.1: Boxplot of participant performances with LEARNED and STATIC in four-room domain



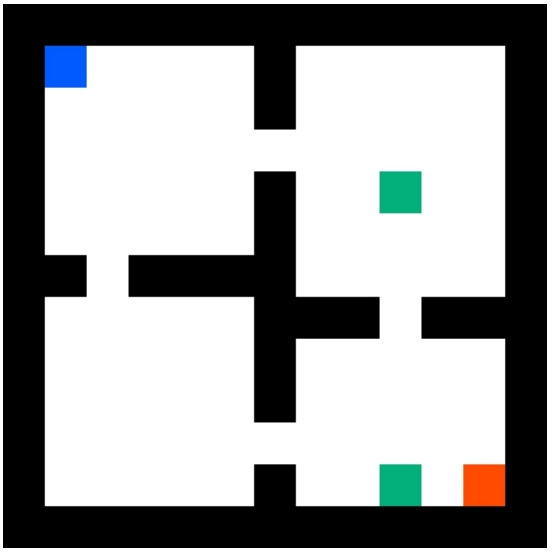
(A): Best subgoal sequence (from P9) with LEARNED.



(B): Worst subgoal sequence (from P3) with LEARNED.



(C): Best subgoal sequence (from P6) with STATIC.



(D): Worst subgoal sequence (from P0) with STATIC.

FIGURE 7.2: Best and worst subgoal sequences acquired from participants in four-room domain. Blue square is start state and red square is goal state. Subgoal closer to start state is younger order.

performance of STATIC was better than that of LEARNED (15518 < 17803 steps) but the worst performance of STATIC was worse than LEARNED (25468 > 23340 steps).

As shown in Figure 7.2, the subgoals in the best subgoal sequence for LEARNED and STATIC were placed in hallways. Both the best subgoal sequences in Figures 7.2(A) and 7.2(C) were in the upper-right hallways. The subgoals in the worst subgoal sequence with LEARNED and STATIC were placed in the same cells but different participants gave them. They were placed near the center of a room. The four-room domain had several optimal trajectories, which had a length of 19 steps. The subgoals given by

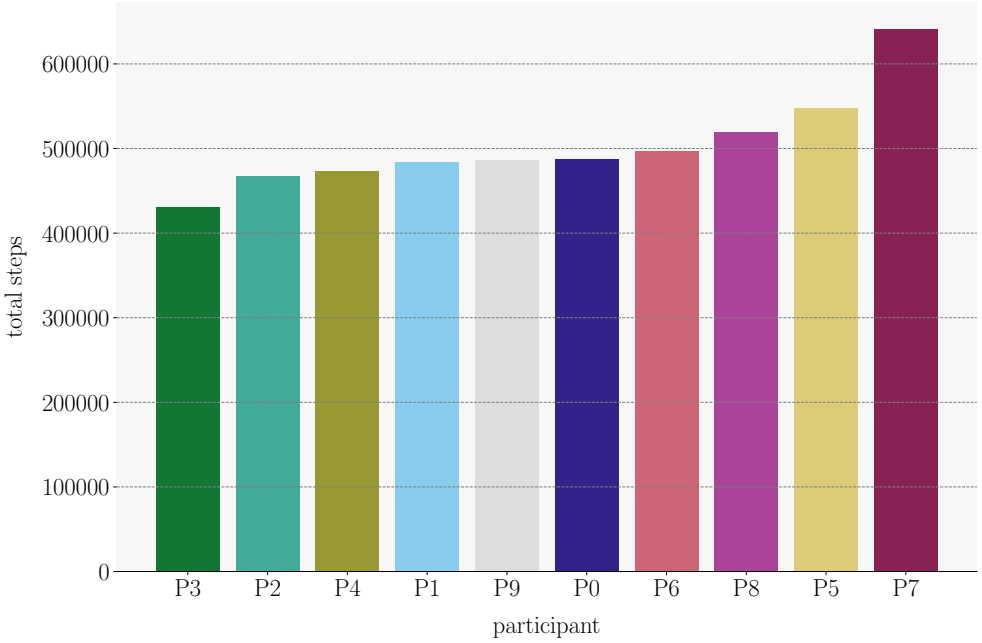
participants could be on optimal trajectories, but the hallways were on more optimal trajectories. The optimal trajectories could be classified into upper-right and bottom-right routes. If an agent selected the upper-right route, it had to visit the upper-right hallways, but it must not visit the near center of a room. The states on many optimal trajectories might match the subgoal-based reward shaping as subgoals.

Figure 7.3 shows bar plots of LEARNED and STATIC among the participants in the four-room and pinball domain. We used the number of steps taken until the goal position as the performance metric and summed the number of steps until the 250th episode. A lower value was better which means that the learning was efficient. The better subgoal sequence given by a participant was plotted to the left in the figure.

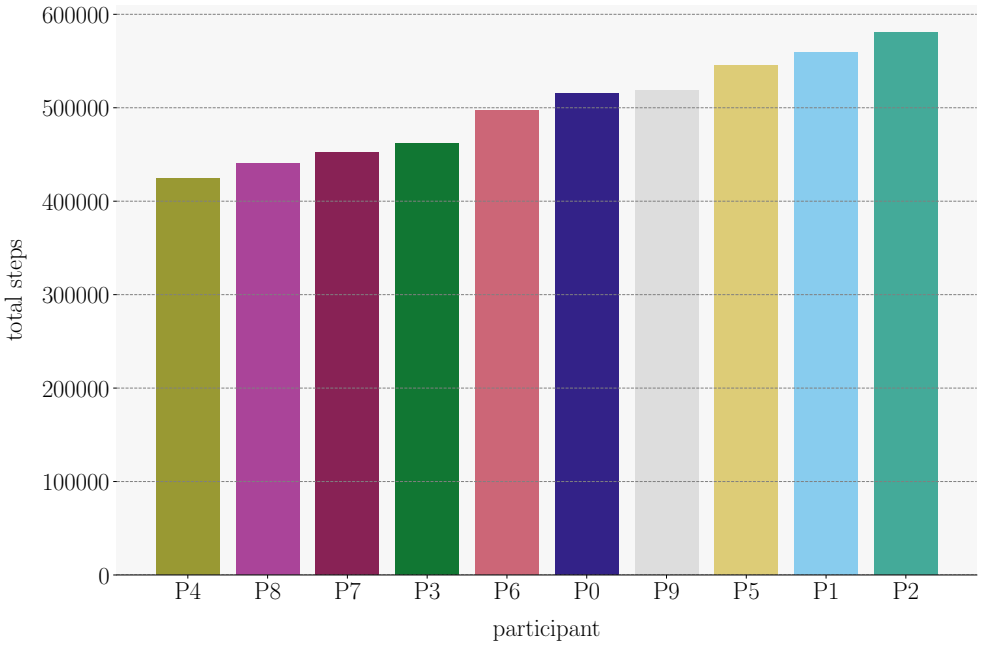
The best subgoal sequence was from P3 and the worst one was from P7, as shown in Figure 7.3(A). The subgoal sequence from P4 was the best and one from P2 was the worst. The difference between the best and worst performances for LEARNED was 210011 steps, and the one for STATIC was 156,435 steps. LEARNED had a larger difference than STATIC. Figure 7.4 shows the best and worst subgoal sequences acquired from participants in the pinball domain.

The best and worst subgoal sequences seemed to be on a similar trajectory as shown in Figure 7.4. The best subgoal sequence of LEARNED and worst subgoal sequence of STATIC were similar. The worst subgoal sequence of LEARNED and best subgoal sequence of STATIC were also similar. The near subgoals to the start position as shown in Figures 7.4(A) and 7.4(A) were placed on the junction connecting a road from the left and the right. Each subgoal near the start position as shown in Figures 7.4(B) and 7.4(A) was placed on the entrance of the first narrow road. Each subgoal near the goal position as shown in Figures 7.4(B) and 7.4(C) was placed in the junction in front of the goal position, and each position was similar. Each subgoal near the goal position as shown in Figure 7.4(B) and 7.4(C) was placed on a narrow road. As shown in Figures 7.4(A) and 7.4(B), subgoals should be placed on a junction but not at an entrance of a narrow road for LEARNED. An agent reached the junctions easier than the entrances. The easier it is for the agent to reach the subgoals, the more LEARNED learns its potential. LEARNED cannot generate the shaping reward if it does not learn its potential. A subgoal with high reachability seems to be important for LEARNED.

The best subgoals of STATIC were similar to the worst of LEARNED. The subgoals were placed on narrow roads and consisted only of three-dimensional positions and did not have three-dimensional velocities. The entrance and junction might allow the agent to move far from the goal. STATIC generated the shaping reward from the early stage of learning. The agent first moved with high randomness and tended to move far from the

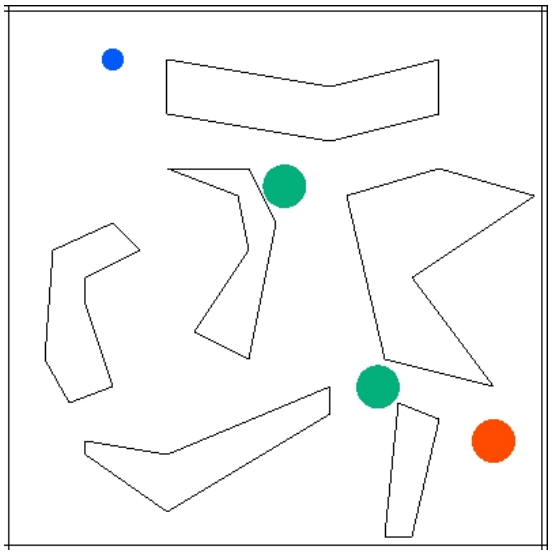


(A): LEARNED.

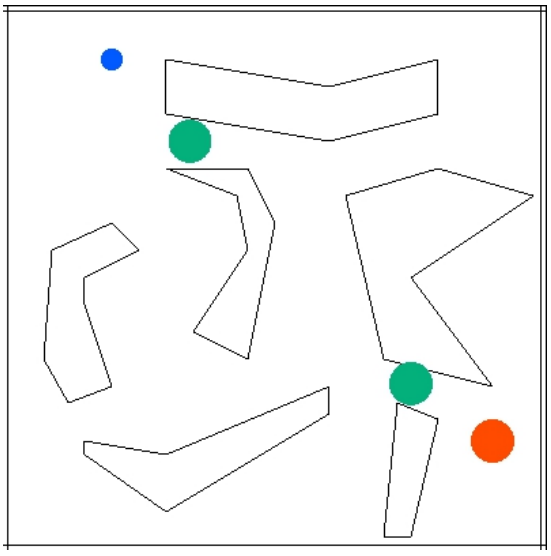


(B): STATIC.

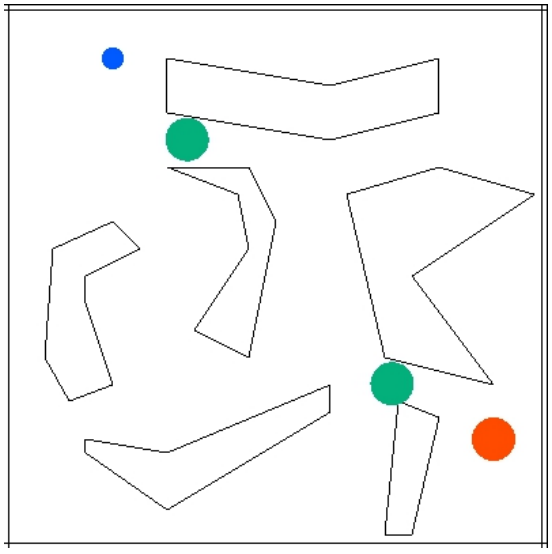
FIGURE 7.3: Bar plot of performances of LEARNED and STATIC among participants in pinball domain



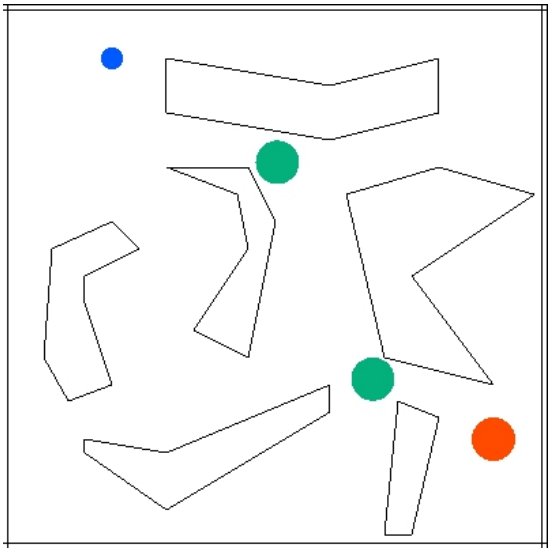
(A): Best subgoal sequence (from P3) with LEARNED.



(B): Worst subgoal sequence (from P7) with LEARNED.



(C): Best subgoal sequence (from P4) with STATIC.

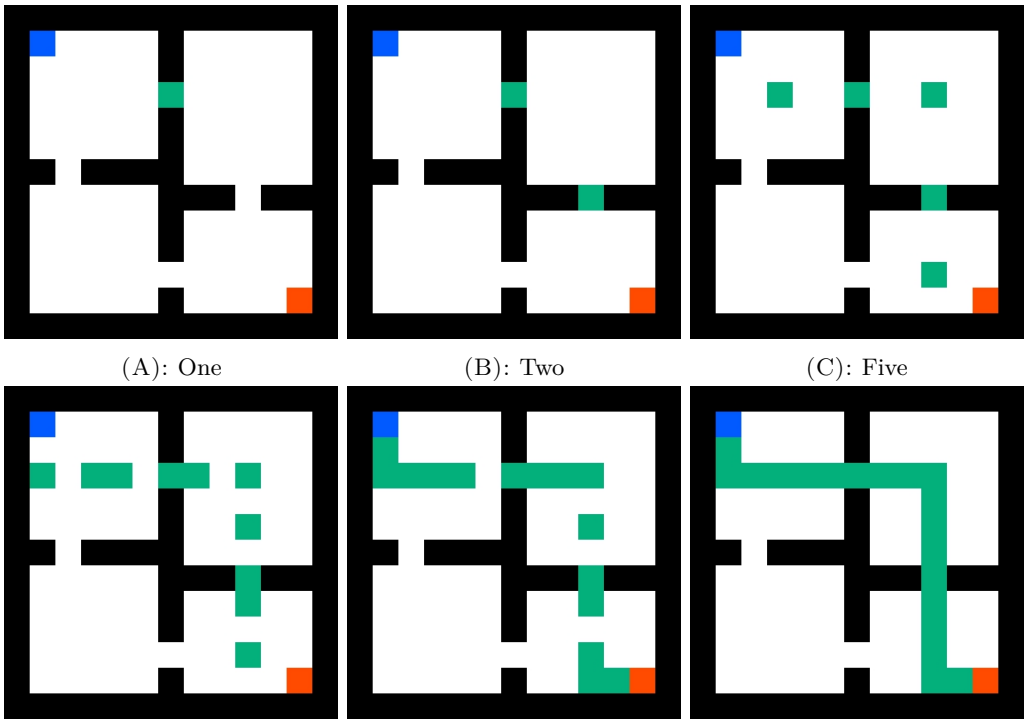


(D): Worst subgoal sequence (from P2) with STATIC.

FIGURE 7.4: Best and worst subgoal sequences acquired from participants in pinball domain. Blue circle is ball placed in start position and red circle is goal area. Subgoal area closer to start position is younger order.

goal. The entrance and junction might motivate such movement. LEARNED generated only low rewards in the early stage of learning because it did not learn sufficient potential.

From the above discussion, LEARNED is suitable for subgoals that are easy to access and included in more optimal trajectories, and STATIC is suitable for subgoals that do not allow the agent to move far from the goal. If the participants know these conditions, their subgoals will further improve the baseline algorithm in learning efficiency.



(A): One (B): Two (C): Five

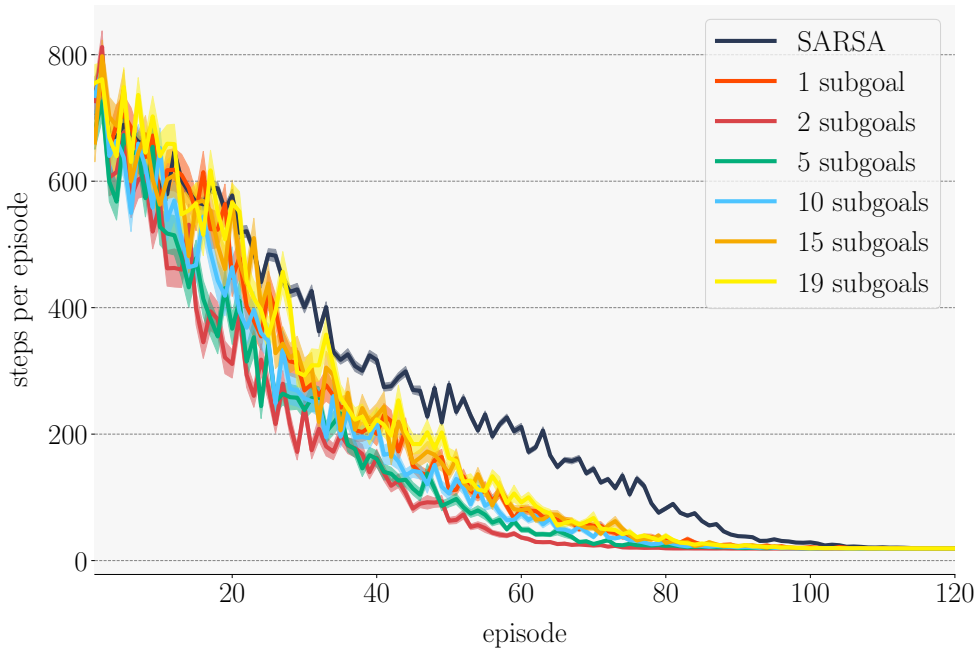
(D): Ten (E): Fifteen (F): Nineteen

FIGURE 7.5: Visual examples of every subgoal

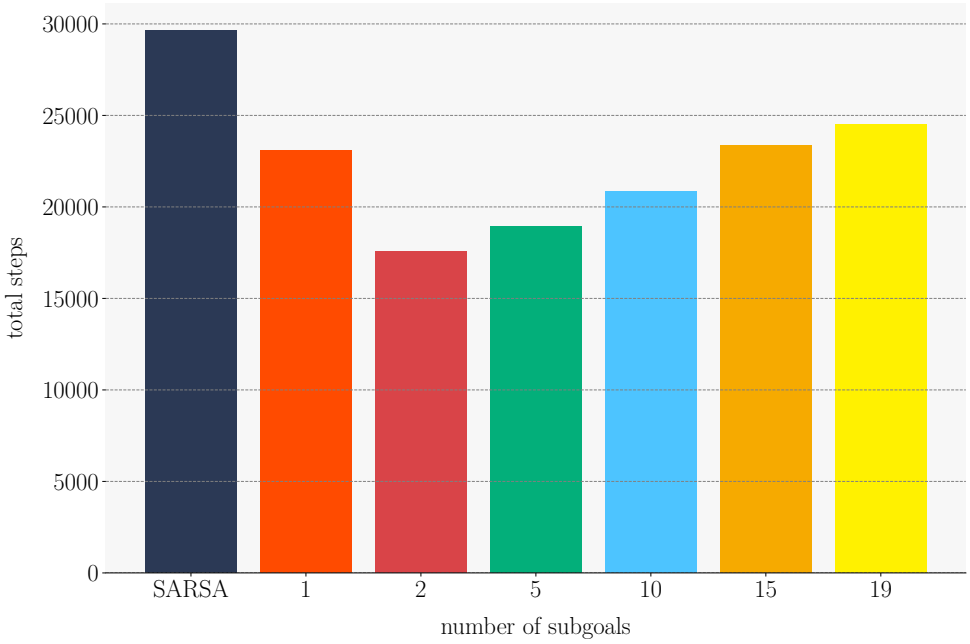
7.1.3 Number of Subgoals

We analyzed how learning efficiency changes due to the number of subgoals. We fixed the number of subgoals to 2, as mentioned in Chapter 6. We increased the number of subgoals to 19 in the four-room domain. The four-room domain has a discrete state space, so the number of states in an optimal trajectory can be determined. The optimal trajectory of the four-room domain includes 19 states except for the start and goal states. We assumed that a subgoal can be placed on the optimal trajectory, and the maximum number of subgoals is 19. We analyzed learning efficiency with 1, 5, 10, 15, and 19 subgoals. The minimum and maximum number of subgoals were 1 and 19, and the first, second, and third quartiles were 5, 10, and 15. We gave subgoals at sequence with a lower number for the near start state. We increased the subgoals by evenly converting a state between start, goal, and subgoals to a subgoal. Figure 7.5 shows the visual examples of every subgoal.

We compared the number of subgoals only in LEARNED because STATIC requires time-consuming hyperparameter tuning per number of subgoals. We used the same algorithmic and environmental settings as those mentioned in Chapter 6. Figure 7.6 shows the learning curves and bar plots among the different numbers of subgoals. We summed the steps until the 120th episode. The figure plots the fewer number of subgoals to the left. SARSA used no subgoals, and it is plotted at left-most.



(A): Learning curves.



(B): Bar plots.

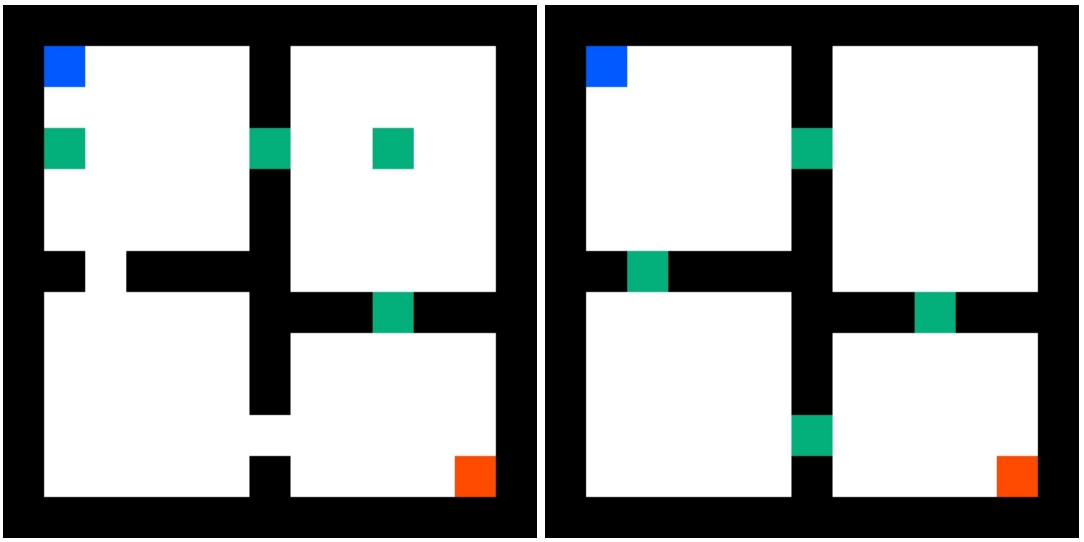
FIGURE 7.6: Boxplot of performances of LEARNED every subgoal in pinball domain

As shown in Figure 7.6, two subgoals were the best, and five subgoals were the second best. Nineteen subgoals were the worst, and 15 were the second worst. All subgoals with LEARNED could outperform SARSA. More subgoals lead to lower performance with more than two subgoals. Too many or too few subgoals deteriorate LEARNED, and there is the best number of subgoals, which was two in the four-room domain. LEARNED can make the baseline algorithm efficient independent of the number of subgoals if subgoals are on an optimal trajectory. The four-room domain has multiple optimal trajectories classified into right-above and left-bottom routes. The two subgoals are on the cells that cover all right-above optimal trajectories. We consider that this is why the two subgoals are the best performance. The single subgoal has the same optimal trajectories as the two ones. After the agent achieves the first subgoal of two, the next subgoal encourages it to reach the goal state. The single subgoal only encourages the agent to achieve the first subgoal and is worse than the two. It is difficult for the agent to achieve more than three subgoals. Our implementation disallows the agent to break the order of subgoals. The agent cannot achieve the second subgoal before achieving the first subgoal. More than five subgoals are more difficult to be achieved in sequence in the four-room domain. The optimal number of subgoals should depend on a domain.

7.1.4 Totally and Partially Ordered

This section describes the difference in the performance of LEARNED between totally ordered and partially ordered subgoal sequences. We used only the totally ordered subgoal sequence mentioned in Chapter 6. A user may want to give a partially ordered subgoal sequence in a task with optimal trajectories. We analyzed the subgoal-based reward shaping with the partially ordered subgoal sequence in the four-room domain. We placed the partially ordered subgoal sequence in the upper-right and bottom-left hallways. The upper-right and bottom-left trajectories required 19 transitions and were optimal. For a fair comparison between totally ordered and partially ordered subgoal sequences, we prepared two totally ordered subgoal sequences, one has two subgoals included in the partially ordered subgoal sequence to align the positions of subgoals as shown in Figure 7.5(B), and the other had the same number of subgoals to the partially ordered subgoal sequence, i.e., four. Figure 7.7 shows the partially ordered subgoal sequence and the totally ordered subgoal sequence with four subgoals we used.

We conducted learning with LEARNED by using the subgoal sequences shown in Figure 7.7. STATIC requires time-consuming hyperparameter tuning so we did not use it. The algorithmic and environmental settings were the same as those mentioned in



(A): Totally ordered subgoal sequence. (B): Partially ordered subgoal sequence.

FIGURE 7.7: Totally and partially ordered subgoal sequences

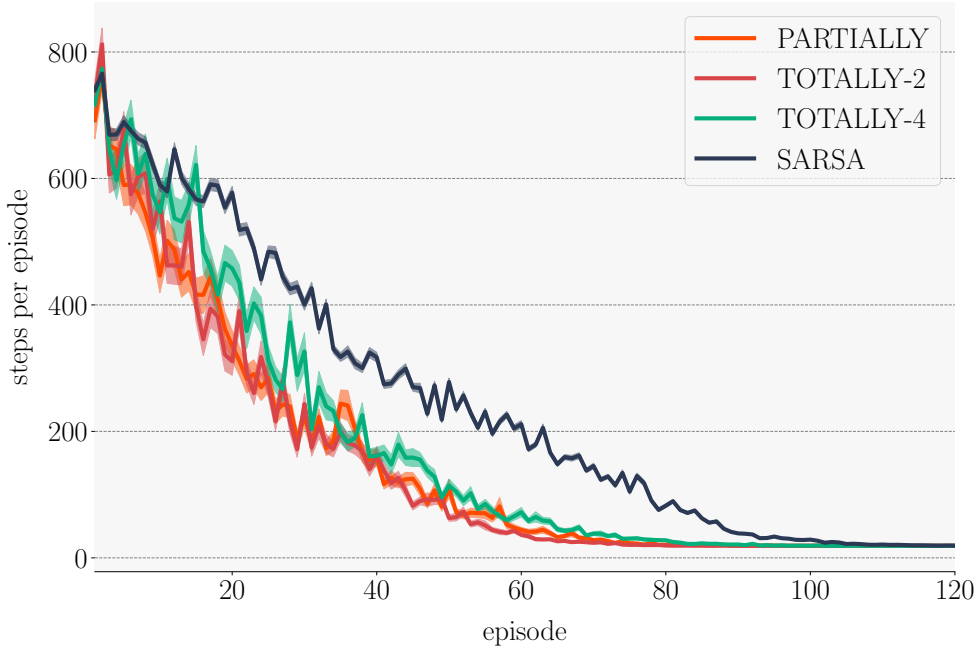


FIGURE 7.8: Learning curves for evaluation of partially ordered subgoals

Chapter 6. We modified the algorithm of LEARNED to incorporate the partially ordered subgoal sequence into it. Figure 7.8 shows the learning curves with a comparison between partially ordered and totally ordered subgoal sequences.

As shown in Figure 7.8, the partially ordered subgoal sequence had similar learning efficiency to the totally ordered subgoal sequence with two subgoals. The totally ordered

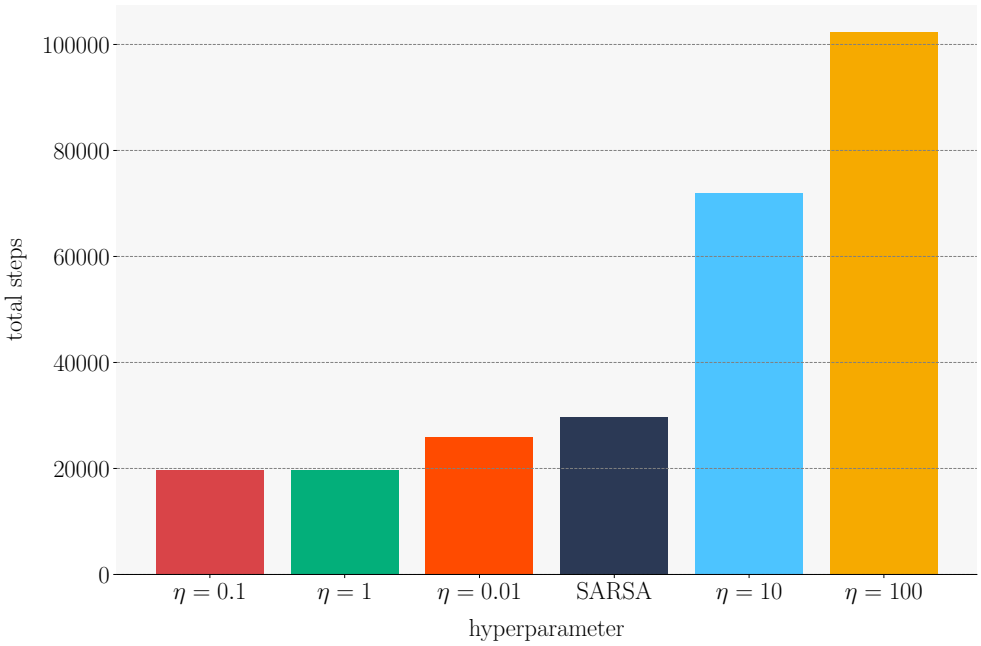
subgoal sequence with four subgoals was worse than the other algorithms with a subgoal sequence. All algorithms with a subgoal sequence performed better than SARSA. LEARNED with a partially ordered subgoal sequence could improve the baseline algorithm in terms of learning efficiency.

7.2 Hyperparameters for Subgoal-based Reward Shaping

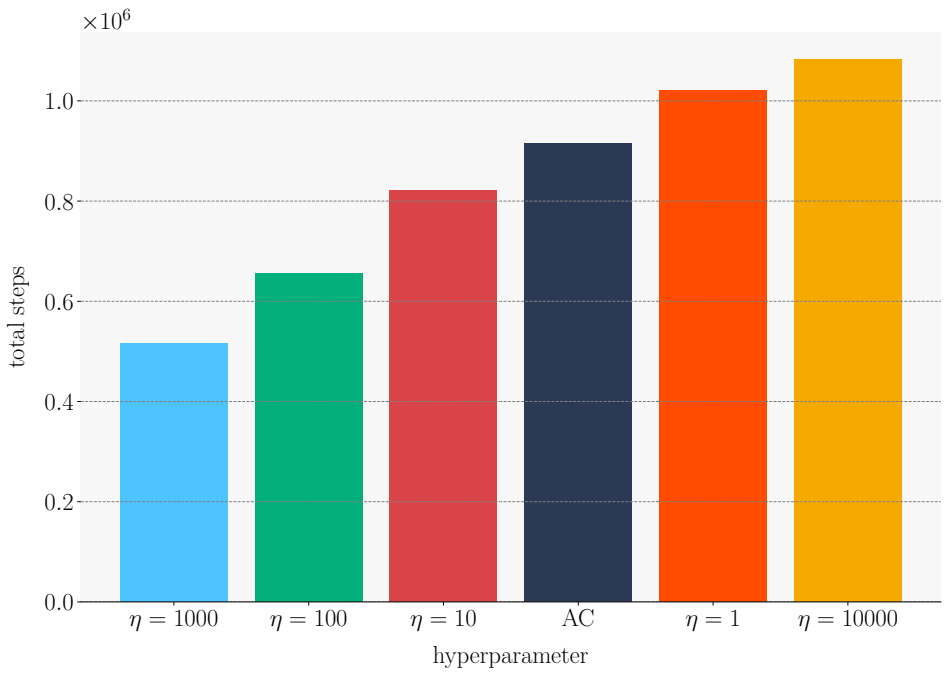
7.2.1 Hyper Parameter for Static Goal-oriented Potential

We analyzed how a hyperparameter of STATIC affects learning efficiency. STATIC has a hyperparameter η which controls the increment of value for the potential. We observed the performances when we changed η and fixed the environmental and algorithmic settings. Figure 7.9 shows the results of a grid search described in Section 6.3. In the grid search, we used only a single participant's subgoal sequence which was randomly selected and fixed the other hyperparameters. The bar plot was calculated by summing steps until the 120th episode for the four-room domain and the 250th episode for the pinball domain. Lower steps show a better performance.

Figure 7.9(A) had similar total steps of $\eta = 0.1$ and $\eta = 1$ in the four-room domain. $\eta = 0.1$ was the best and $\eta = 100$ was the worst. The difference between the best and the worst was 82,719 steps. $\eta = 0.01$, $\eta = 0.1$, and $\eta = 1$ were better than SARSA, and $\eta = 10$ and $\eta = 100$ were worse than SARSA. Figure 7.9(B) shows $\eta = 1000$ as the best and $\eta = 10000$ as the worst. $\eta = 10$, $\eta = 100$, and $\eta = 1000$ were better than SARSA, and $\eta = 1$ and $\eta = 10000$ were worse. The difference between the best and the worst was 567454 steps. As seen in Figure 7.9, STATIC was highly sensitive to the hyperparameter η. A wrong η such as $\eta = 10$ and $\eta = 100$ in the four-room domain could deteriorate the performance of the method. This may incur a cost for tuning hyperparameters. The environmental rewards of goals were 1 and 10,000 for four-room and pinball, respectively. The bigger η than the environmental rewards deteriorated the baseline algorithm in terms of learning efficiency. The results of η with the same size as the reward were different, and it improved the baseline in the four-room domain and deteriorated the baseline in the pinball domain. The four-room domain had a shorter optimal trajectory than the pinball domain, and it may lead to this result. Though η with the one-thousandth reward improved the baseline, the ten-thousandth made learning slow. From these results, the bigger η than the environmental reward might deteriorate the baseline, and the smaller and not too small η might improve the baseline.



(A): Four-rooms domain.



(B): Pinball domain.

FIGURE 7.9: Results of grid searches.

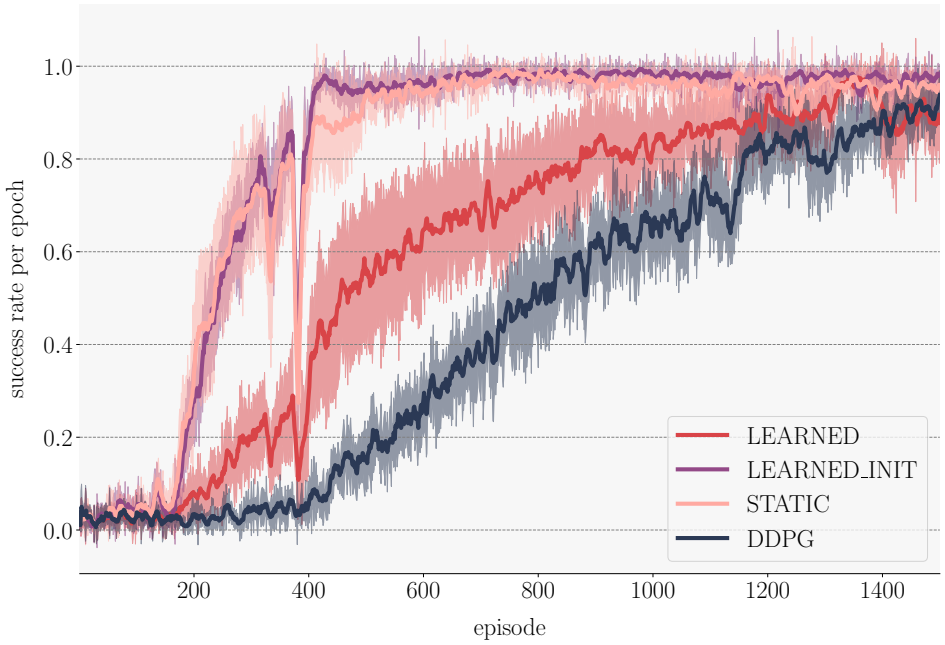


FIGURE 7.10: Learning curves of DDPG, STATIC, LEARNED, and LEARNED INIT.

7.2.2 Initial Values for Learned Potential

This section reports how the initial values of LEARNED affect its performance. We set the initial values of LEARNED to zero, as mentioned in Chapter 6. We analyzed its performance in the pick-and-place domain because the domain revealed that STATIC performed more effectively than LEARNED compared with the other domains. The hyperparameter η for STATIC was set to 1, and we set the initial values following STATIC, which increased the potential per subgoal achievement by 1. We conducted the learning experiment in the same setting as that mentioned in Section 6.5 to evaluate LEARNED set with the initial values. Figure 7.10 shows the learning curves of DDPG and LEARNED with and without value initialization (LEARNED INIT).

LEARNED with the value initialization had a similar performance to STATIC, and highly improved LEARNED without the value initialization. A user can set the initial values of LEARNED to improve its learning efficiency if the user knows a good potential function. Empirically, a good potential is an optimal value function [7], which is unknown, so the user may estimate it from the reward function, discount factor, and length of an optimal trajectory. However, it is too difficult to estimate it in certain situations. The user needs to use LEARNED and search its initial values if the search is reasonable; otherwise, the user trains LEARNED with zero-initialization.

Section 6.5 showed that LEARNED performed worse than STATIC. The value of an abstract state closer to a goal state was lower than an abstract state farther from the goal state in the early stage of learning. This might be why LEARNED could not make learning efficient without initialization. STATIC always had a larger potential in a closer abstract state because it fixed its potential by the hyperparameter. The inversion of potentials provokes the negative shaping reward to the transition for the goal. The update of potential with the constraint where the closer state has a larger potential might solve this problem. This is because the constraint can avoid the negative shaping reward.

7.3 Types of Reward Functions

Chapter 6 reports the experimental results in environments with a positive goal reward. This section analyzed the performance of subgoal-based reward shaping in the environment with two types of rewards. The first type of reward generated a negative reward every step and zero rewards in a goal state. The second type had mixed positive and negative rewards.

7.3.1 Step Penalized Rewards

We changed the reward function that outputs a negative reward at every step and zero rewards in a goal state. We used the four-room domain because it was the simplest of the three domains and easy to analyze. The settings were the same as those mentioned in Section 6.3 except for the rewards. We compared LEARNED with two potentials with SARSA. We used the two totally ordered subgoals shown in Figure 7.5(B). Figure 7.11 shows the learning curves in the step-penalized rewards. Lines and shaded areas show mean steps and standard errors of steps.

As shown in Figure 7.11, STATIC and SARSA had almost the same performance, and LEARNED was the best until about the 100th episode and then became the worst after that. In contrast to the positive goal reward mentioned in Section 6.3, LEARNED and STATIC could not perform better than SARSA in terms of the step-penalized rewards. Grzes and Kudenko stated that positive potentials could not improve the baseline much and negative potentials made it difficult to adjust hyperparameters to meet their conditions with negative step rewards [36]. Their conditions depended on the discount factor and size of potential. STATIC was positive, and LEARNED was negative because of learning from environmental rewards. LEARNED and STATIC could not perform better than SARSA, which agreed with Grzes and Kudenko's results.

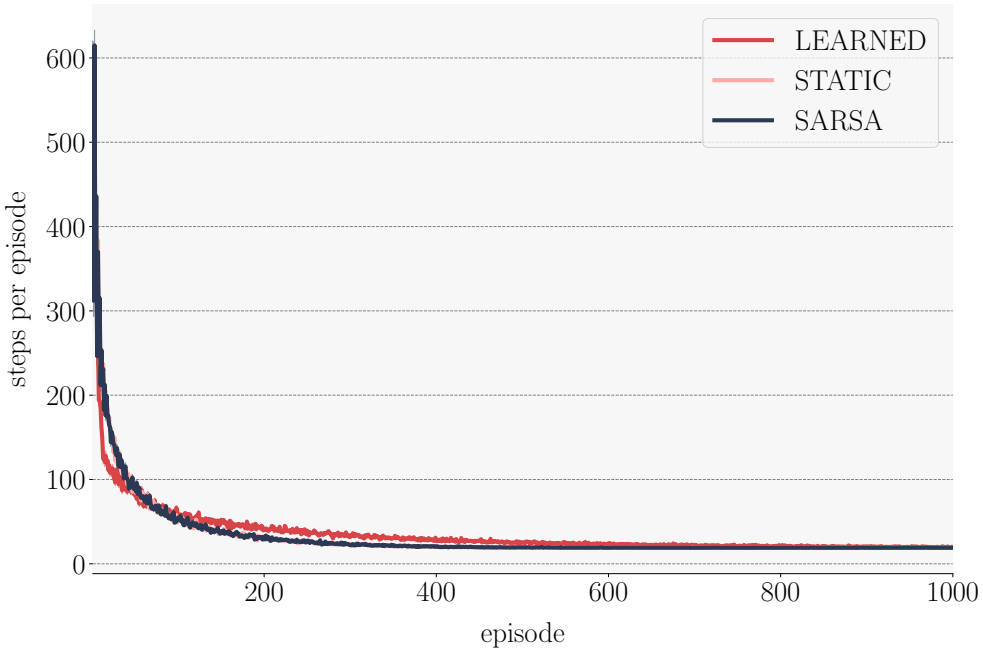


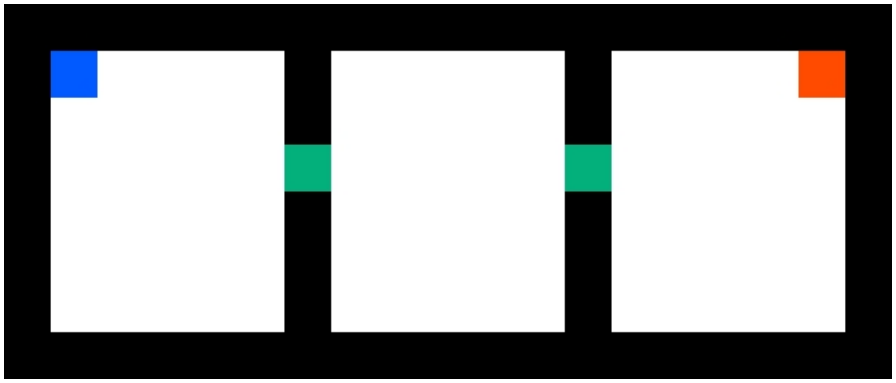
FIGURE 7.11: learning curves in step-penalized rewards.

We might have found that the hyperparameters could meet their conditions, but it was too time-consuming and we would need to modify certain parts of their equations for our settings. LEARNED could and STATIC could not perform well with negative step rewards, and this remains an open problem.

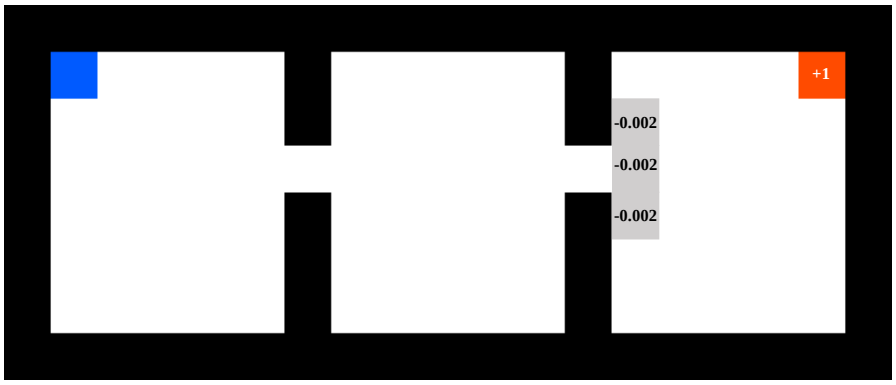
7.3.2 Mixed Positive and Negative Rewards

We consider mixed positive and negative rewards to reveal the strength of LEARNED compared with STATIC. The mixed rewards had a large positive goal reward of +1 and small negative step rewards of -0.002 because subgoal-based reward shaping can work well in a positive goal reward, and many negative rewards might make learning worse. We used the navigation task in a three-room domain, which had two hallways connecting three rooms. An agent must visit the two hallways to reach a goal state. We used two hallways as the totally ordered subgoal sequence with two subgoals. Figure 7.12 illustrates the subgoal sequence and rewards in the three-room domain.

Figure 7.12 shows the start position in blue, goal position in red, subgoals in green, and texts explaining the rewards. We aligned the other settings with those mentioned in Section 6.3. We compared LEARNED, STATIC and SARSA. Figure 7.13 shows the learning curves in the three-room domain with mixed positive and negative rewards.



(A): Subgoal sequence.



(B): Rewards.

FIGURE 7.12: Three-room domain.

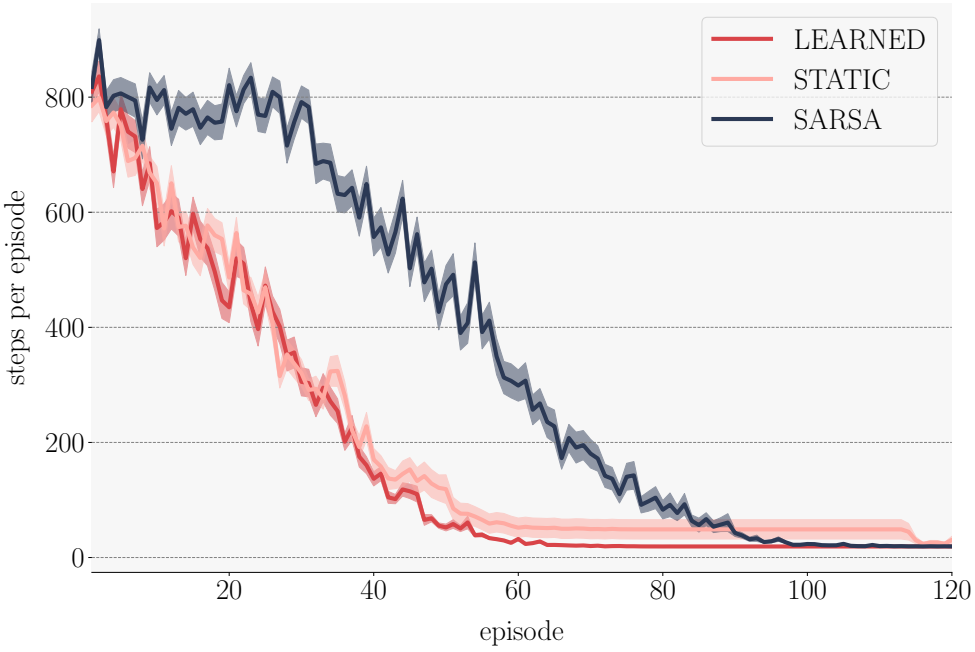


FIGURE 7.13: learning curves in mixed positive and negative rewards.

As shown in Figure 7.13, LEARNED and STATIC learned faster than SARSA through almost all learning stages, but STATIC seemed to be asymptotic to non-optimal values in the 60th episode. SARSA was finally faster regarding asymptotic to optimal values than STATIC. The standard error of STATIC was the largest of the three algorithms. This is because both successes and failures of episodes existed for STATIC. With STATIC, it is assumed that the potential increases with the same amount for every subgoal achievement. Small penalized rewards go against this assumption because they decreased the value after a second subgoal achievement in the third room. LEARNED can adjust the potential from experiences, so it performed well even with small penalized rewards.

7.4 Limitations

This section reports the limitations of subgoal-based reward shaping. Subgoal-based reward shaping aims to make reinforcement learning efficient by using provided subgoals. The improvement of learning efficiency is required to spread reinforcement learning in industry and daily lives because it speeds up development and decreases its cost, so it is necessary for almost situations. We consider two bottlenecks to using subgoal-based reward shaping. Section 7.4.1 reported the availability of a sequence of subgoals as the first bottleneck. The second bottleneck written in Section 7.4.2 is the type of environment. Section 7.4.3 reported the limitations regarding the teaching of subgoals.

7.4.1 Difficulties of acquiring subgoals

We implicitly assumed that a human can set subgoals more easily than acquiring the optimal trajectories from a start state to a goal state, which is conventional human knowledge that improves the efficiency of reinforcement learning with IRL. We need to verify this assumption by conducting experiments with participants to measure the cognitive load in setting various subgoals and trajectories.

The ease of acquiring subgoals may depend on the kind of task. We think that there are two steps that a human can take to set subgoals in a task. At first, the human comes up with an optimal trajectory for a task. The optimal trajectory includes subgoals. If the human cannot come up with this trajectory, he or she cannot know the subgoals. In the second, the human selects states as subgoals in a trajectory. The trajectory includes states, and humans must prioritize them because the high-prioritized states become subgoals. A task in which a human can process these two steps easily would be an appropriate one for subgoals. We consider that board games with an opponent such as chess, shogi and Go [74] have difficulty in the first step. The agent has to change his/her

strategy depending on the opponent in these games, and a teacher cannot come up with an optimal trajectory before the play starts. Even if the optimal trajectory was found, the teacher might be difficult to choose a subgoal from the optimal trajectory in tasks whose states are similar or hard to be recognized by the teacher. The suitable states are easy to be recognized and prioritized by the teacher. A teacher's observation need not be the same as a state as long as the teacher can choose subgoals, so we can transform the state into an observation that a teacher is easy to recognize. we can restore the state if we keep relationships between a state and an observation.

We plan to conduct a large user study to make clear what the best task for acquiring subgoals is. The viewpoint of the human teacher may be a factor. For this, we can consider a bird's-eye view and a first-person view as levels. The navigation tasks in this paper involved a bird's-eye view. An example of a task with a first-person view is a first-person shooter game such as ViZDoom [75]. We hypothesize that a task with a bird's-eye view is more suitable for providing subgoals than one with a first-person view. Another factor may be the explainability of the optimal trajectories. The trajectory of the four-room domain is easy for a human teacher to explain, such as explaining that the agent should first go to the top-right hallway and then the bottom-right hallway. The teacher can select subgoals easily in this domain like the top-right and bottom-right hallways. In the Hand Manipulation task [23], it is hard for the teacher to explain how the robotic hand of the agent should move. It may also be difficult for the teacher to select subgoals. These are related to explicit and implicit knowledge, respectively. We hypothesize that a task including an optimal trajectory that the teacher can explain is more suitable than one in which the teacher cannot explain the trajectory under subgoal settings. Future work is also to test these hypotheses in an extensive user study.

7.4.2 Type of Environment

We summarize the environment for reinforcement learning in which subgoal-based reward shaping can improve the baseline algorithm. The environment in Chapter 6 had three types of state-action pairs: discrete action and discrete state, discrete actions and continuous states, and continuous action and continuous state in the environment. We used separate baseline methods including a table form and approximate value function. The subgoal-based reward shaping could improve the baseline algorithm for all types of state-action pairs. It did not depend on the type of stat-action pair and the baseline algorithm. In addition, it could perform efficiently with a high-dimensional observation. The four-room and pinball domains have an absorbing state that ends an episode when an agent reaches a goal, and the pick-and-place task continues after reaching the goal. Many robotic tasks do not have an absorbing state or a termination condition [76] and

continue up to a predefined step. Subgoal-based reward shaping makes learning efficient regardless of whether there is an absorbing state or not. This depends on the environmental reward function. Section 7.3.1 shows it cannot improve the baseline algorithm with negative step rewards; in comparison, Chapter 6 shows its effectiveness with a positive goal reward. The learned potential can learn efficiently with a positive goal reward and some small negative rewards as in Section 7.3.2. Subgoal-based reward shaping performs efficiently only with a positive goal reward. However, negative step rewards can be converted into a single positive goal reward. Negative step rewards decrease the value of the state that the agent has visited once. The agent tends to visit a non-visited state that has a high value and explores more widely than within a single positive goal. Though an agent explores efficiently within negative step rewards, exploration bonus methods [77] can eliminate the strength within a positive goal reward. An exploration bonus outputs a high additional reward in a low frequency state. We used Random Network Distillation, which is one of the exploration bonus methods in Section 6.5. Subgoal-based reward shaping could work well with the exploration bonus. Therefore, this limitation is not crucial. The start and goal positions were fixed in Chapter 6 because appropriate subgoal sequences could be changed and acquiring a subgoal sequence became more expensive. The positions of the start and goal decide where the subgoals are. For example, in the four-room domain, the hallway connecting the above-left and the above-right rooms is not a subgoal if the start is placed in the above-right room. The agent should not visit the hallway to reach the goal position. The hallway is a good subgoal if the start is in the left-above room as shown in Section 6.3. Therefore, a participant must provide subgoal sequences for every pair of start and goal positions because the subgoals are varied by the pair of start and goal positions. We consider that subgoal-based reward shaping would not work well in an environment with various start and goal positions such as Multi-Goal Reinforcement Learning [23]. We need to add a method for selecting a sequence of subgoals from a start-goal position pair to work well in multi-goal reinforcement learning. After selecting the sequence of subgoals, the problem is the same as the case of a single start-goal position. The agent can access the start position in the first step of an episode, and the goal position is accessible in multi-goal reinforcement learning [23]. The method of selecting the sequence of subgoals only has relations between a start-goal position pair and a sequence of subgoals if the state space is discrete such as in the four-room domain. A data-driven function may be required if the state space is continuous because there are too many start-goal position pairs to make the relations. We should collect many records storing start-goal position pairs and sequences of subgoals from many participants. Section 7.4.3 discussed the cognitive load required from participants to acquire many records.

7.4.3 Teaching Knowledge of Subgoals

We now mention the difficulties when a human teacher provides the sequence of subgoals. LEARNED requires a sequence of subgoals in each task. A human teacher can provide this, and it is easier to acquire from a human teacher and with better quality than from automatic generation in many situations. The difficulties to acquire subgoals might be classified into two types. The first difficulty is how the system acquires subgoals from a human teacher, and the second is how the human teacher should provide them.

We consider system-sided difficulty. It is easy to acquire subgoals from humans in certain domains such as the four-room and pinball domains, but not easier in others such as the pick-and-place domain. The easy domains can give a teacher almost the entire structure of states. We gave the participants a bird's-eye view of the positions of the start, goal, and obstacles, as well as those of subgoals. The pick-and-place domain, however, had an object that moved during a task. Therefore, participants could not provide subgoals after they observed the initial positions of the object. This is an example in which the almost entire structure of state space is difficult to show. The interactive RL [78] approach might alleviate this difficulty. With this approach, an agent acts by following its policy, and a human teacher gives feedback to it. The agent then updates its policy from the feedback in an iteration [79]. Christiano at el. proposed that a human teacher selects a preferable trajectory in a pair of trajectories, and a policy update uses the preferable one [80]. The trajectory can be interpreted as a query to know the entire structure of state space because it includes a sequence of states. The human teacher can know the structure and give subgoals simultaneously if he/she provides the subgoal as feedback from the trajectory. Applying interactive RL to acquire subgoals is future work.

We also consider the human teacher-sided difficulty. We had no established method of providing subgoals, which may depend on individual intuitive considerations. For the four-room domain, as shown in Figure 5.3, almost all the subgoals were located within the right-top and right-bottom rooms. From this, we think that many participants tended to consider the right-down path as the shortest one. This may mean humans abstractly have a standard methodology and preference for giving subgoals. The exciting observation was that half of the participants set a subgoal in the hallways.

For the pinball domain, as shown in Figure 5.4, the subgoals were scattered at the gap between objects and the branch points. They seem to be at the change points of actions on an optimal trajectory. We can consider that repeated actions are composed of a macro-action. This is a similar approach to [81].

For the pick-and-place task, five participants provided the same subgoals. The way to provide subgoals was different from the other tasks. We used the descriptive answer-type form. These results almost show the tendency for people to provide similar subgoals exists.

We consider that people change their strategy for providing subgoals in response to the task properties. We showed that subgoals on an optimal trajectory are useful for subgoal-based reward shaping in Section 7.1.2. Subgoal-based reward shaping with the participants' subgoals could improve the baseline RL algorithm as mentioned in Chapter 6, so the participants might have been able to provide subgoals placed on near-optimal trajectories. We told the participants what the target of the task was and that every task might be easy to imagine on an optimal trajectory. A human teacher might be able to provide useful subgoals for subgoal-based reward shaping if he/she understands the optimal trajectory of the task. A system designer only instructs the target of a task, enabling a human teacher to understand an optimal trajectory without detailed methodology; thus, useful subgoals can be acquired.

Chapter 8

Conclusion

Learning a policy is time-consuming in reinforcement learning. We aim for accelerating learning with reward transformation based on human subgoal knowledge. The difficulty of integrating subgoal knowledge into the potential was a problem in improving learning efficiency. We proposed subgoal-based reward shaping based on potential-based reward shaping and two types of potential functions for it. The first potential function is the static goal-oriented potential and has a hyperparameter controlling the increment of potential. We proved that an extension for using a subgoal sequence of potential-based reward shaping also guaranteed policy invariance. The static goal-oriented potential overcame the difficulty of integrating subgoal knowledge into the potential. The second potential function is the learned potential, and it learns its potential. The learned potential does not require a hyperparameter to shape the potential, and it reduces the cost of development. We collected subgoal sequences from participants in a user study. The distribution of subgoals that participants provided was biased more than random-generated subgoal sequences. Experiments showed that subgoal-based reward shaping improves the baseline reinforcement learning algorithm in the middle of learning. Each participant's subgoal sequence was more useful for subgoal-based reward shaping to improve in learning efficiency than each random-generated subgoal sequence. We discussed that most random-generated subgoals are on near-optimal trajectories, and this led to a small difference between participants' and random-generated subgoal sequences. We analyzed the subgoal quality among the participants and found that subgoals included in the best subgoal sequence were on many optimal trajectories. There was an appropriate number of subgoals for subgoal-based reward shaping. Though too many or

too few subgoals were not more effective, subgoal-based reward shaping improved the baseline algorithm as long as the subgoal was on the optimal trajectory. The partially ordered subgoal sequence was useful as well as the totally ordered subgoal sequence. A hyperparameter analysis showed that the static goal-oriented subgoals had a sensitive hyperparameter, and the initialization of values improved the learned potential. We applied subgoal-based reward shaping to a variety of rewards. Negative step rewards were not effective, but mixed positive and negative rewards were effective. Moreover, the learned potential is superior to the static goal-oriented potential.

This dissertation did not include a user interface and method for interacting with the user. How the user can easily provide subgoals is future work. Subgoal-based reward shaping is limited to working well with a positive goal reward, but many types of reward such as negative step rewards can be transformed into a positive reward, so this is a small limitation. When many people provide subgoals, the lack of a methodology might be a limitation. Subgoal-based reward shaping could improve the baseline algorithm as long as the user provides subgoals that are on optimal trajectories. A detailed methodology might be not required, and the user might be able to collect useful subgoals.

Reinforcement learning has a larger potential to enrich people's lives than ever. Accelerating reinforcement learning significantly decreases the time and cost of development and might increase the number of applications. We should utilize human knowledge as much as possible for acceleration. Subgoals are a useful choice and often easy to use. We believe that subgoal-based reward shaping encourages people to use subgoals that have never been used and is helpful for accelerating reinforcement learning.

Bibliography

[1] Matthew E. Taylor. "Improving Reinforcement Learning with Human Input". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*. International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 5724–5728.

[2] Zhaodong Wang and Matthew E. Taylor. "Effective Transfer via Demonstrations in Reinforcement Learning: A Preliminary Study". In: *Proceedings of the AAAI 2016 Spring Symposium*. CEUR-WS.org, 2016, pp. 77–83.

[3] Shane Griffith et al. "Policy Shaping: Integrating Human Feedback with Reinforcement Learning". In: *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2013, pp. 2625–2633.

[4] Yusuf Aytar et al. "Playing Hard Exploration Games by Watching YouTube". In: *Advances in Neural Information Processing Systems 32*. Curran Associates Inc., 2018, pp. 2935–2945.

[5] Andrew Y. Ng and Stuart J. Russell. "Algorithms for Inverse Reinforcement Learning". In: *Proceedings of the 17th International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 2000, pp. 663–670.

[6] Jonathan Ho and Stefano Ermon. "Generative Adversarial Imitation Learning". In: *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 2016, pp. 4565–4573.

[7] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping". In: *Proceedings of the 16th International Conference on Machine Learning (ICML 1999)*. Morgan Kaufmann Publishers Inc., 1999, pp. 278–287.

[8] Sam Devlin and Daniel Kudenko. "Dynamic Potential-based Reward Shaping". In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*. International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 433–440.

[9] Ariel Rosenfeld, Matthew E. Taylor, and Sarit Kraus. "Leveraging Human Knowledge in Tabular Reinforcement Learning: A Study of Human Subjects". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2017, pp. 3823–3830.

[10] Saleema Amershi et al. "Power to the People: The Role of Humans in Interactive Machine Learning". In: *AI Mag.* 35.4 (2014), pp. 105–120.

[11] Saurabh Arora and Prashant Doshi. "A survey of inverse reinforcement learning: Challenges, methods and progress". In: *Artificial Intelligence* 297 (2021), p. 103500.

[12] Dario Amodei et al. "Concrete Problems in AI Safety". In: *CoRR* abs/1606.06565 (2016). arXiv: 1606.06565.

[13] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. USA: Prentice Hall Press, 2009. ISBN: 0136042597.

[14] Paul F Christiano et al. "Deep Reinforcement Learning from Human Preferences". In: *Advances in Neural Information Processing Systems 30* (2017), pp. 4299–4307.

[15] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 627–635.

[16] Francisco Cruz et al. "Improving interactive reinforcement learning: What makes a good teacher?" In: *Connection Science* 30.3 (2018), pp. 306–325.

[17] W. Bradley Knox and Peter Stone. "Interactively Shaping Agents via Human Reinforcement: The TAMER Framework". In: *The 15th International Conference on Knowledge Capture*. 2009.

[18] Richard S. Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artificial Intelligence* 112 (1999), pp. 181–211.

[19] Pierre-Luc Bacon, Jean Harb, and Doina Precup. "The Option-critic Architecture". In: *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI 2017)*. AAAI Press, 2017, pp. 1726–1734.

[20] Alexander Sasha Vezhnevets et al. "FeUdal Networks for Hierarchical Reinforcement Learning". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. JMLR.org, 2017, pp. 3540–3549.

[21] Yuuji Ichisugi et al. "Hierarchical Reinforcement Learning with Unlimited Recursive Subroutine Calls". In: *Artificial Neural Networks and Machine Learning (ICANN 2019)*. Springer International Publishing, 2019, pp. 103–114.

[22] Junichi Murata, Yasuomi Abe, and Keisuke Ota. "Introduction and Control of Subgoals in Reinforcement Learning". In: *Proceedings of the 25th Conference on Proceedings of the 25th IASTED International Multi-Conference: Artificial Intelligence and Applications*. AIAP'07. Innsbruck, Austria: ACTA Press, 2007, pp. 329–334.

[23] Matthias Plappert et al. "Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research". In: *CoRR* abs/1802.09464 (2018). arXiv: 1802.09464.

[24] Tom Schaul et al. "Universal Value Function Approximators". In: *Proceedings of The 32nd International Conference on Machine Learning* (2015), pp. 1312–1320.

[25] Ashvin Nair et al. "Visual Reinforcement Learning with Imagined Goals". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio et al. 2018, pp. 9209–9220.

[26] Soroush Nasiriany et al. "Planning with goal-conditioned policies". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., Dec. 2019, pp. 14843–14854.

[27] Marcin Andrychowicz et al. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems 30* (2017), pp. 5048–5058.

[28] Siyuan Li et al. "Learning Subgoal Representations with Slow Dynamics". In: *International Conference on Learning Representations*. 2021.

[29] Ofir Marom and Benjamin Rosman. "Belief reward shaping in reinforcement learning". In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*. AAAI press, 2018, pp. 3762–3769. ISBN: 9781577358008.

[30] Alper Demir, Erkin Çilden, and Faruk Polat. "Landmark Based Reward Shaping in Reinforcement Learning with Hidden States". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2019)*. International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 1922–1924.

[31] Yang Gao and Francesca Toni. "Potential Based Reward Shaping for Hierarchical Reinforcement Learning". In: *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, 2015, pp. 3504–3510.

[32] Siyuan Li et al. "Hierarchical Reinforcement Learning with Advantage-Based Auxiliary Rewards". In: *Advances in Neural Information Processing Systems 32* 32 (2019), pp. 1409–1419.

[33] Sujoy Paul, Jeroen Vanbaar, and Amit Roy-Chowdhury. "Learning from Trajectories via Subgoal Discovery". In: *Advances in Neural Information Processing Systems 32* (2019), pp. 8411–8421.

[34] Eric Wiewiora, Garrison Cottrell, and Charles Elkan. "Principled Methods for Advising Reinforcement Learning Agents". In: *Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003)*. Morgan Kaufmann Publishers Inc., 2003, pp. 792–799.

[35] Anna Harutyunyan et al. "Expressing Arbitrary Reward Functions as Potential-Based Advice". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI Press, 2015, pp. 2652–2658. ISBN: 0262511290.

[36] Marek Grzes and Daniel Kudenko. "Theoretical and Empirical Analysis of Reward Shaping in Reinforcement Learning". In: *2009 International Conference on Machine Learning and Applications*. Dec. 2009, pp. 337–344.

[37] Adam Laud and Gerald DeJong. "The Influence of Reward on the Speed of Reinforcement Learning: An Analysis of Shaping". In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. ICML'03. Washington, DC, USA: AAAI Press, 2003, pp. 440–447.

[38] Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010.

[39] Richard S Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems 12*. MIT Press, 2000, pp. 1057–1063.

[40] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018.

[41] Vijay R. Konda and John N. Tsitsiklis. "Actor-Critic Algorithms". In: *Advances in Neural Information Processing Systems 12*. MIT Press, 2000, pp. 1008–1014.

[42] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016.

[43] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013).

[44] David Silver et al. "Deterministic Policy Gradient Algorithms". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, 22–24 Jun 2014, pp. 387–395.

[45] Eric Wiewiora. "Potential-Based Shaping and Q-Value Initialization are Equivalent". In: *Journal of Artificial Intelligence Research (JAIR)* 19 (2003), pp. 205–208.

[46] Marek Grzes and Daniel Kudenko. "Multigrid Reinforcement Learning with Reward Shaping". In: *Artificial Neural Networks - ICANN 2008 , 18th International Conference, Prague, Czech Republic, September 3-6, 2008, Proceedings, Part I*. Ed. by Vera Kurková, Roman Neruda, and Jan Koutnı´k. Vol. 5163. Lecture Notes in Computer Science. Springer, 2008, pp. 357–366.

[47] Marek Grzes and Daniel Kudenko. "Online learning of shaping rewards in reinforcement learning". In: *Neural networks* 23 (2010), pp. 541–550.

[48] Ishai Menache, Shie Mannor, and Nahum Shimkin. "Q-Cut - Dynamic Discovery of Sub-goals in Reinforcement Learning". In: *Proceedings of the 13th European Conference on Machine Learning (ECML 2002)*. Springer-Verlag, 2002, pp. 295–306.

[49] Marco Wiering and Jurgen Schmidhuber. *HQ-Learning: discovering markovian subgoals for non-markovian reinforcement learning*. Tech. rep. IDSIA, Jan. 1996, pp. 1–13.

[50] Bram Bakker and J Schmidhuber. "Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization". In: *Proceedings of the 8-th Conference on Intelligent Autonomous Systems (IAS 2004)*. 2004, pp. 438–445.

[51] Özgür Şimşek and Andrew G Barto. "Using relative novelty to identify useful temporal abstractions in reinforcement learning". In: *Twenty-first international conference on Machine learning - ICML '04*. Vol. 91. New York, New York, USA: ACM Press, 2004, p. 95.

[52] Özgür Şimşek, Alicia P Wolfe, and Andrew G Barto. "Identifying useful subgoals in reinforcement learning by local graph partitioning". In: Association for Computing Machinery (ACM), Feb. 2006, pp. 816–823.

[53] Sandeep Goel and Manfred Huber. "Subgoal Discovery for Hiearchical Reinforcement Learning Using Learned Policies". In: *Proceedings of the 16th International FLAIRS Conference* (2003), pp. 346–350.

[54] Shie Mannor et al. "Dynamic Abstraction in Reinforcement Learning via Clustering". In: *Proceedings of the 21st International Conference on Machine Learning (ICML 2004)*. Morgan Kaufmann Publishers Inc., 2004, pp. 71–78.

[55] Richard Catrambone. "Subgoal Learning". In: *Encyclopedia of the Sciences of Learning*. Ed. by Norbert M Seel. Boston, MA: Springer US, 2012, pp. 3230–3233.

[56] Richard Catrambone. "Improving examples to improve transfer to novel problems". In: *Memory & Congnition* 22.5 (Sept. 1994), pp. 606–615.

[57] Sarah Weir et al. "Learnersourcing Subgoal Labels for How-to Videos". In: *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. CSCW '15. Vancouver, BC, Canada: Association for Computing Machinery, Feb. 2015, pp. 405–416.

[58] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362.

[59] Matthew E. Taylor and Peter Stone. "Transfer Learning for Reinforcement Learning Domains: A Survey". In: *Journal of Machine Learning Research* 10 (2009), pp. 1633–1685.

[60] Matthew E. Taylor and Peter Stone. "Cross-Domain Transfer for Reinforcement Learning". In: *Proceedings of the 24th International Conference on Machine Learning* (2007).

[61] Satoshi Tanaka. "js-STAR 2012". In: *http://www.kisnet.or.jp/nappa/software/star/* (2012).

[62] G.D. Konidaris, S. Osentoski, and P.S. Thomas. "Value Function Approximation in Reinforcement Learning using the Fourier Basis". In: *Proceedings of the 25nd AAAI Conference on Artificial Intelligence (AAAI 2011)*. AAAI Press, 2011, pp. 380–385.

[63] Greg Brockman et al. *OpenAI Gym*. 2016.

[64] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard, Version 2.2*. Specification. Sept. 2009.

[65] Prafulla Dhariwal et al. *OpenAI Baselines*. 2017.

[66] Yuri Burda et al. "Exploration by random network distillation". In: *International Conference on Learning Representations*. 2019.

[67] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[68] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[69] Abien Fred Agarap. "Deep Learning using Rectified Linear Units (ReLU)". In: *CoRR* abs/1803.08375 (2018). arXiv: 1803.08375.

[70] B. T. Polyak and A. B. Juditsky. "Acceleration of Stochastic Approximation by Averaging". In: *SIAM Journal on Control Optimization* 30.4 (July 1992), pp. 838–855.

[71] James Bergstra, Daniel Yamins, and David Cox. "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research. Atlanta, Georgia, USA: PMLR, 2013, pp. 115–123.

[72] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, July 2019, pp. 2623–2631.

[73] Lajanugen Logeswaran et al. "Few-shot Subgoal Planning with Language Models". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, United States: Association for Computational Linguistics, July 2022, pp. 5493–5506.

[74] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144.

[75] Michal Kempka et al. "ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning". In: *CoRR* abs/1605.02097 (2016). arXiv: 1605.02097.

[76] Yan Duan et al. "Benchmarking deep reinforcement learning for continuous control". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, June 2016, pp. 1329–1338.

[77] Deepak Pathak et al. "Curiosity-driven exploration by self-supervised prediction". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, Aug. 2017, pp. 2778–2787.

[78] Christian Arzate Cruz and Takeo Igarashi. "A Survey on Interactive Reinforcement Learning: Design Principles and Open Challenges". In: *Proceedings of the 2020 ACM Designing Interactive Systems Conference*. New York, NY, USA: Association for Computing Machinery, July 2020, pp. 1195–1209.

[79] W Bradley Knox and Peter Stone. "Interactively shaping agents via human reinforcement: the TAMER framework". In: *Proceedings of the fifth international conference on Knowledge capture*. K-CAP '09. Redondo Beach, California, USA: Association for Computing Machinery, Sept. 2009, pp. 9–16.

[80] Paul F Christiano et al. "Deep Reinforcement Learning from Human Preferences". In: *Advances in Neural Information Processing Systems 30*. Ed. by I Guyon et al. Curran Associates, Inc., 2017, pp. 4299–4307.

[81] Aravind S. Lakshminarayanan, Sahil Sharma, and Balaraman Ravindran. "Dynamic Action Repetition for Deep Reinforcement Learning". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI'17. San Francisco, California, USA: AAAI Press, 2017, pp. 2133–2139.

List of Publication

Journal

[1] Takato Okudo and Seiji Yamada. "Learning Potential in Subgoal-Based Reward Shaping". In: IEEE Access 11 (2023), pp.17116–17137. doi: 10.1109/ACCESS.2023.3246267. url: https://doi.org/10.1109/ACCESS.2023.3246267.

[2] Takato Okudo and Seiji Yamada. "Subgoal-Based Reward Shaping to Improve Efficiency in Reinforcement Learning". In: IEEE Access 9 (2021), pp.97557–97568. doi: 10.1109/ACCESS.2021.3090364. url: https://doi.org/10.1109/ACCESS.2021.3090364.

Conference (peer-reviewed)

[1] Takato Okudo and Seiji Yamada. "Online Learning of Shaping Reward with Subgoal Knowledge". In: AAMAS' 21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021. Ed. by Frank Dignum et al. ACM, 2021, pp.1613–1615. doi: 10.5555/3463952.3464177. url: https://www.ifaamas.org/Proceedings/aamas2021/pdfs/p1613.pdf.

[2] Takato Okudo and Seiji Yamada."Reward Shaping with Dynamic Trajectory Aggregation". In: International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021. IEEE, 2021, pp.1–9. doi: 10.1109/IJCNN52387.2021.9533401. url: https://doi.org/10.1109/IJCNN52387.2021.9533401.96

Conference (not peer-reviewed)

[1] 奥戸嵩登 and 山田誠二. "階層型強化学習における人間のサブゴール知識転移". In: 人工知能学会全国大会論文集 JSAI2019 (2019), 1Q2J202–1Q2J202. doi:10.11517/pjsai.JSAI2019.0_1Q2J202.97