Reinforcement Learning in Multi-dimensional State-Action Space using Random Rectangular Coarse Coding and Gibbs Sampling

Hajime Kimura

***Abstract*— This paper presents a coarse coding technique and an action selection scheme for reinforcement learning (RL) in multi-dimensional and continuous state-action spaces following conventional and sound RL manners. RL in high-dimensional continuous domains includes two issues: One is a generalization problem for value-function approximation, and the other is a sampling problem for action selection over multi-dimensional continuous action spaces. The proposed method combines random rectangular coarse coding with an action selection scheme using Gibbs-sampling. The random rectangular coarse coding is very simple and quite suited both to approximate Q-functions in high-dimensional spaces and to execute Gibbs sampling. Gibbs sampling enables us to execute action selection following Boltsmann distribution over high-dimensional action space. The algorithm is demonstrated through Rod in maze problem and a redundant-arm reaching task comparing with conventional regular grid approaches.**

I. INTRODUCTION

Reinforcement learning (RL) is a promising paradigm for robots to acquire control rules automatically in unknown environments. The most popular RL algorithm is tabular Q-learning [13] because of its simplicity and well-developed theory. However, RL in real world applications is to deal with high-dimensional continuous state-action spaces, that immediately raises *the curse of dimensionality* [6], in which costs increase exponentially with the number of the state-action variables. It includes two issues in RL: One is a generalization problem for Q-function over high-dimensional continuous spaces, and the other is a sampling problem that is to select actions following a given distribution over high-dimensional continuous action spaces.

To deal with continuous state space, a number of RL algorithms combining with function approximation techniques have been invented: *Q-net* [5] is an extended Q-learning algorithm by means of using neural networks to generalize the Q-function, *SARSA with CMAC* [10] is a better method because it is based on linear function approximation, that has good theoretical convergence properties in RL [1]. [9] and [12] proposed RL algorithms with instance-based function approximation, and [7] extended it to consider continuous action spaces. However, these works are not aimed at handling high-dimensional spaces. [11] introduced *hashing* to cope with high-dimensional spaces. Hashing is a consistent pseudo-random collapsing of a large tiling into a much smaller set of tiles. This approach is quite favorable, however, the action sampling problem remains.

Dept. of Marine Engineering, Graduate School of Engineering, Kyushu University, 744 Motooka, Nishi-ku, Fukuoka, 819-0395 JAPAN
kimura@nams.kyushu-u.ac.jp

In the Q-learning, the state-action value function directly approximates the optimum state-action value function independent of the policy being followed. During learning, there is a difficult exploration vs exploitation trade-off to be made [4]. One standard practice is known as *Boltzmann exploration*, however, its naive computational cost increases exponentially with the number of action variables, independently of the generalization costs. One solution to overcome this problem is to use *Gibbs sampling*, that is a Markov chain Monte Carlo (MCMC) method for drawing samples in high-dimensional spaces [3][8].

This paper presents a new approach combining random rectangular coarse coding with an action selection scheme using Gibbs-sampling. The proposed coding method is quite suited both to approximate Q-functions in high-dimensional spaces, and to execute Gibbs sampling.

II. PROBLEM FORMULATION

Let $\mathcal{S}$ denote state space, $\mathcal{A}$ be action space, $\mathcal{R}$ be a set of real number. At each discrete time t, the agent observes state $s_t \in \mathcal{S}$, selects action $a_t \in \mathcal{A}$, and then receives an instantaneous reward $r_t \in \mathcal{R}$ resulting from state transition in the environment. In Markov decision processes (MDPs), the reward and the next state may be random, but their probability distributions are assumed to depend only on s_t and a_t. And the next state s_{t+1} is chosen according to the transition probability $T(s_t, a, s_{t+1})$, and the reward r_t is given randomly following the expectation $r(s_t, a)$. The agent does not know $T(s_t, a, s_{t+1})$ and $r(s_t, a)$ ahead of time. The objective of RL is to construct a policy that maximizes the agent's performance. Consider cumulative discounted reward as a performance measure for infinite horizon tasks:

$$V_t = \sum_{k=0}^{\infty} \gamma^k \, r_{t+k} \ , \tag{1}$$

where the discount factor, $0 \leq \gamma \leq 1$ specifies the importance of future rewards, and V_t is the value at time t. In MDPs, the value can be defined as:

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t \, r_t \middle| \, s_0 = s, \pi \right] \ , \tag{2}$$

where $E\{\cdot\}$ denotes the expectation. The learning objective in MDPs is to find optimum policies that maximize the value of each state s defined by (2). The state $\mathcal{S}$ and the action $\mathcal{A}$ are both defined as multi-dimensional continuous spaces in this paper.

III. Random Rectangular Coarse Coding for High-dimensional Spaces

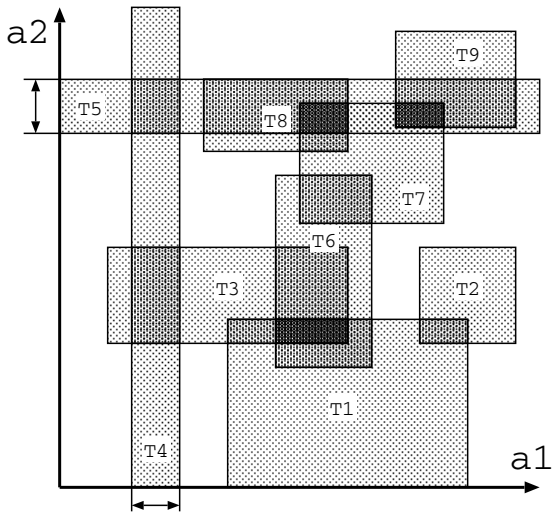

Fig. 1. An example of the random rectangular coarse coding using 9 features in two-dimensional action space. The coarse feature T4 is sensitive to the component a1, and it is defined by the arrow width in the sub-space a1. But the T4 ignores the component a2, then the rectangle covers the other all sub-space a2. The feature T5 is defined similarly.

Value function approximation can be significantly enhanced through the use of feature extraction, a process that maps the input space x into some vector $F(x) = (f_1(x), f_2(x), \cdots, f_n(x))$, that is refered to *feature vector* associated with x. Feature vectors summarize what are considered to be important characteristics of the input space. One of the most important cases of the value function approximation is that in which the approximate function is a linear function of the parameter vector $W = (w_1, w_2, \cdots, w_n)$ that has the same number of the components as the feature vector $F(x)$. For example, the state value is approximated by

$$V(x) = \sum_{i=1}^{n} f_i(x)\, w_i \quad . \tag{3}$$

This value function is said to be *linear*. When the set of the feature vectors $\{F(x)|x \in \mathcal{S}\}$ is *linearly independent*, then it is known that the predictions of the TD-methods converge in expected value to the ideal predictions [11].

To generate feature vectors that are linearly independent in high-dimensional spaces, random rectangular coarse coding is described below. This idea is directly inspired from "*CMAC with hashing*" [11]. And it is also derived from consideration of the Parti-game algorithm [6] in multi-joint arm problems, in which the complexity of the ultimate task remains roughly constant as the number of degrees of freedom increases. These deliberation lead that a fine state-action representation may be practicable by regarding some subsets of the state (or action) variables in high-dimensional spaces.

To explain the concept of the random rectangular coding, consider a randomized rectangular feature $f(x)$ that is an element of the feature vector. When an input vector x is inside of the rectangle, then the corresponding feature has the value 1, otherwise 0. In our approach, the feature rectangles are overlapped in many places. This kind of approach is known as *coarse coding* [11]. In higher dimensional space, a feature $f(x)$ selects a few arbitrary sensitive components of the input vector x, and composes a hyper rectangle in the sensitive subspace. That is, a feature $f(x)$ ignores non-sensitive components of the input vector x. For example in Fig.1, the input vector x is two-dimensional (a_1, a_2), and the rectangular T1, T2, $\cdots$ T9 are associated with components of the feature vector respectively. Notice that T1, T2, T3, T6, T7, T8 and T9 are sensitive to the components both a_1 and a_2, however, T4 is sensitive only to a_1 and T5 is sensitive only to a_2. In this paper, the set of sensitive components, the ranges and the size of the rectangles are given randomly, but it is also possible to consider some other sophisticated methods. Aggregating a number of such features, it is possible to give a set of feature vectors that is linearly independent so that the state-action values can be adequately approximated over essential states (or actions) in real-world tasks, without exponential growth of the memory requirements.

In this paper, state-features $f_j(s)$ where $j \in \{1, 2, \ldots T_s\}$ that are defined in state space and action-features $g_k(a)$ where $k \in \{1, 2, \ldots T_a\}$ that are defined in action space are introduced. It is desirable that L_1 norm of the feature vectors are constant (1 is the best[1]) in linear function approximation. Therefore, state-feature vectors $F(s)$ that has $2T_s$ components $F(s) = (F_1(s), F_2(s), \cdots F_{2T_s}(s))$ and action-feature vectors $G(a)$ that has $2T_a$ components $G(a) = (G_1(a), G_2(a), \cdots G_{2T_a}(a))$ are given by the state-features $f_j(s)$ and action-features $g_k(a)$ respectively according to

$$F_j(s) = \begin{cases} f_j(s) & \text{, where } j \le T_s \\ 1 - f_{(j-T_s)}(s) & \text{otherwise.} \end{cases} \tag{4}$$

$$G_k(a) = \begin{cases} g_k(a) & \text{, where } k \le T_a \\ 1 - g_{(k-T_a)}(a) & \text{otherwise.} \end{cases} \tag{5}$$

Then the approximate value function of state-action pairs (i.e., Q-function) is given by

$$Q(s,a) = \frac{1}{T_s\, T_a} \sum_{j=1}^{2T_s} \sum_{k=1}^{2T_a} F_j(s)\, G_k(a)\, w_{jk} \quad , \tag{6}$$

where w_{jk} are parameters of which the number is $2T_s \times 2T_a$, and $\frac{1}{T_s\, T_a}$ is a normalization factor so that the L_1 norm of the feature vectors composed of $F_j(s)$ and $G_k(a)$ keep 1 constant. The parameters w_{jk} are updated by a gradient-descent method. The specification are shown later. In this paper, the agent selects action following Boltzmann distribution using Q-function. The proposed generalization method has the following advantages:

- The rectangular feature can directly represent mutual dependence between its sensitive components. It is useful to incorporate domain knowledge about the task into the agent.

[1]The reason is to keep consistency of Q-learning's learning rate parameter avoiding influence of magnitude of the norm.

- The computation both to generate feature vectors and to update parameters are quite simple and easy.
- The computation for action selection in high-dimensional action-space can be effective particularly with using Gibbs sampling as shown in the next section.

Although this approximation itself is promising, naive execution of the Boltzmann exploration over high-dimensional continuous action-space gives rise to exponential growth of the computational cost.

IV. Sampling and Learning in High-dimensional Space

A. Conventional Approach: Flat Action Selection

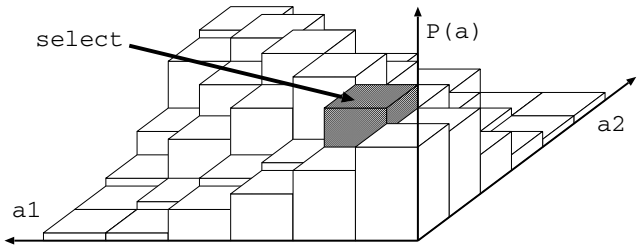

Fig. 2. An action-selection with a regular grid approach in two-dimensional action space. The probability function of the action selection $P(a)$ is represented by quantizing the action space $a1 - a2$ into a finite number of cells.

During learning, Boltzmann exploration is a standard strategy to cope with exploration vs exploitation trade-off in the Q-learning. A common approach to treat multi-dimensional continuous action is to quantize the action space into a finite number of cells and aggregate all actions within each cell. Then the action is treated discretely. Note that each cell needn't have own variables to represent action value, because the value is provided by the coarse coding. In this paper, regular grid partitioning is used that divide D equivalent parts into each dimension of the action-space. That is, when the action-space is N-dimensional, the partitioning yields D^N cells. Each cell is considered to be a discrete action a_i where $i = \{1, \cdots, D^N\}$. For each decision epoch, an action a_i is selected following an associated probability function $P(a_i)$. Fig.2 shows the regular grid partitions and its distribution function in two-dimensional action-space. In this paper, the action-selection probability $P(a_i)$ is given by Boltzmann distribution that contains associated $Q(s, a_i)$ so that

$$P(a_i) = \frac{\exp\left(\frac{Q(s,a_i) - offset(s)}{T \times width(s)}\right)}{\sum_{j=1}^{D^N} \exp\left(\frac{Q(s,a_j) - offset(s)}{T \times width(s)}\right)} , \qquad (7)$$

where T is a temperature parameter that controls exploration behavior. In many cases, $offset(s) = 0$ and $width(s) = 1$, but this paper recommends[2] $offset(s)$ to be average of Q-value in state s and $width(s)$ to be deviation of Q-value in state s. As shown in (7), the naive flat action-selection scheme needs to calculate all discrete action values, that gives rise to exponentially increase with the number of the dimension. For this reason, Q-learning nor SARSA have little opportunity to apply real-world tasks that have high-dimensional continuous action-space.

[2]The reason is to keep consistensy of parameter T avoiding magnitude influence of Q-value.

B. Selecting Action by Gibbs Sampling

This section describes an action-selection method for high-dimensional action space using Gibbs sampling, that is a powerful technique for generating samples from multi-dimensional complex distribution. In the Gibbs sampler, the components of the multidimensional random variable are resampled in turn, conditional on the others, using some fixed ordering. Fig.3 illustrates the Gibbs sampling for a case with two variables $(a1, a2)$. The distribution for action selection probability is equal to Fig.2. In the initial step of Gibbs sampling, an arbitrary sample $(a1, a2)$ is selected in advance. In each iteration, the top right (1) in Fig.3 represents selecting action $a1$ following the conditional probability given $a2$, and the bottom left (2) in Fig.3 represents selecting action $a2$ following the conditional probability given $a1$ at the top right (1) in Fig.3, then the iteration is completed. The bottom right (3) in Fig.3 represents the next iteration, selecting action $a1$ again, following the conditional probability given $a2$ at the bottom left (2) in Fig.3. After a sufficient number of iterations, we can take $a1$ and $a2$ as the sample from the distribution. The resulting sample can be considered to be independent of the sample in the initial step. It is a difficult problem to predict the lower bounds of the number of the iteration, therefore we should choose it experimentally. Gibbs sampling is particularly well-adapted when the joint distribution is not known explicitly, but the conditional distribution of each variable is known. In high-dimensions, getting joint distribution costs too expensive computational resources. For example, if a regular grid partition divides 10 parts into each dimension of the 8-dimensional action space, then finding the joint distribution for the flat action selection costs memory capacity and computation over 10^8 variables, whereas the Gibbs sampling, that uses conditional distribution, costs $10+8$ variables and the number of iteration can be reduced to 10^2 or 10^3. That is, Gibbs sampling cuts down the computational cost over $1/10000$.

Throughout this paper, probability of the action selection is given by (7), but we use particular notation for Gibbs sampling. Suppose that a regular grid partition divides D parts into each dimension of the N-dimensional action space, same as the case of the flat action selection. We give a N-dimensional action a as $a = (a^1, a^2, \ldots a^N)$, where each component a^n ($n \in \{1, 2, \cdots N\}$) is $a^n \in a^n_d$ ($d \in \{1, 2, \cdots D\}$). A value of state s and action a is given by $Q(s, a) = Q(a^1, a^2, \ldots a^N | s)$. This value is also equal to the flat action selection method's. Sampled components of the action vector at the iteration t in the Gibbs sampling is given by $a^1(t), a^2(t), \ldots a^N(t)$. Then, the Gibbs sampler updates components of the action vector at the iteration $t+1$ following the conditional distribution as

$$a^1(t+1) \quad \sim \quad P(a^1 | a^2(t), a^3(t), \ldots a^N(t)) ,$$

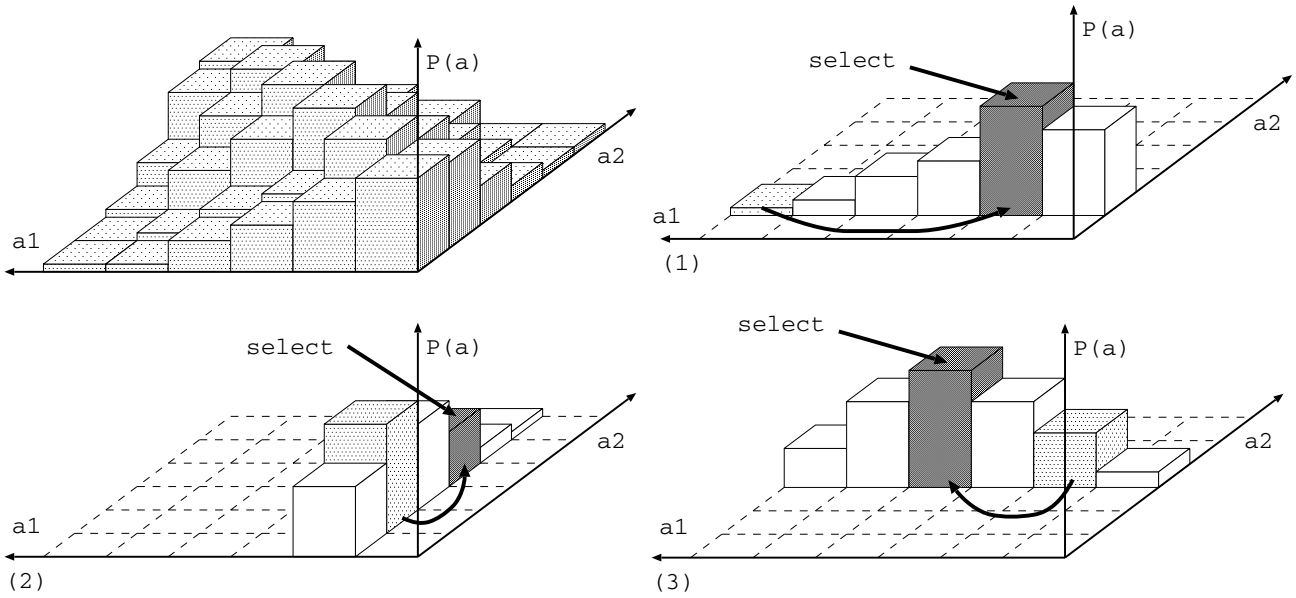

Fig. 3. An example of the Gibbs sampling scheme with the regular grid approach in two-dimensional action space. The top left shows a joint distribution of the action selection probability. The top right (1) represents selecting action $a1$ following the conditional probability given $a2$. The bottom left (2) represents selecting action $a2$ following the conditional probability given $a1$ at the top right (1). The bottom right (3) represents selecting action $a1$ again, following the conditional probability given $a2$ at the bottom left (2).

$$\begin{aligned} a^2(t+1) &\sim P(a^2|a^1(t), a^3(t), \dots a^N(t)) \ , \\ &\vdots \\ a^N(t+1) &\sim P(a^N|a^1(t), a^2(t), \dots a^{N-1}(t)) \ , \end{aligned}$$

where the conditional probability $P(a_i^n|a^1, a^2, \dots a^N)$ is given by the following Boltzmann distribution:

$$P(a_i^n|a^1, a^2, \dots a^N) = \frac{\exp\left(\frac{Q(a^1, a^2, \dots a_i^n, \dots a^N|s) - offset(s)}{T \times width(s)}\right)}{\sum_{d=1}^{D} \exp\left(\frac{Q(a_d^1, a_d^2, \dots a_d^n, \dots a_d^N|s) - offset(s)}{T \times width(s)}\right)} \ . \quad (8)$$

It is simply repetition of similar processes to the flat action selection's shown in (7), however, getting off with one dimensional action selection. The obvious advantage is that the Gibbs sampler can get equivalent[3] samples following (7) without checking out all the Q-values over the high-dimensional action-space. Note that since the calculation of the conditional probability in each iteration is along an arbitrary axis of the action space, the computational cost can be reduced in the proposed rectangular coarse coding.

C. Learning Algorithms

This section describes Q-learning and SARSA algorithms combined with random rectangular coarse-coding and the action selection using Gibbs sampling. As shown in the section III, state-feature vectors $F(s)$ that has $2T_s$ components $F(s) = (F_1(s), F_2(s), \cdots F_{2T_s}(s))$ and action-feature vectors $G(a)$ that has $2T_a$ components $G(a) = (G_1(a), G_2(a), \cdots G_{2T_a}(a))$ are given by the state-features $f_j(s)$ where $j \in \{1, 2, \dots T_s\}$ and action-features $g_k(a)$ where $k \in \{1, 2, \dots T_a\}$ respectively according to (4) and (5). Then the approximate value function is given by (6). The agent observes state s, and selects action a_i following Boltzmann distribution given by (7). However, actual process of choosing action would be executed following Gibbs sampling given by (8). After that, the agent observes the next state s' and instantaneous reward r resulting from the state transition. The standard Q-learning algorithm updates values by

$$\begin{aligned} \text{TD\_error} &= r + \gamma \max_a Q(s', a) - Q(s, a_i) \ , \quad (9) \\ Q(s, a_i) &\leftarrow Q(s, a_i) + \alpha \, \text{TD\_error} \ , \quad (10) \end{aligned}$$

where γ is a discount factor, α is a learning rate parameter, where $0 \leq \alpha \leq 1$. In my approach, the Q-function is represented by (6), therefore the updating of (10) is executed by modifying the parameters w_{jk} using a gradient-descent method as

$$w_{jk} \leftarrow w_{jk} + \frac{F_j(s) \, G_k(a_i)}{T_s \, T_a} \, \alpha \, \text{TD\_error} \ . \quad (11)$$

Unfortunately, it is difficult to operate $\max_a Q(s', a)$ strictly in high-dimensions, we are to substitute approximated $\max_a Q(s', a)$ found in the process of the Gibbs sampling. For the other hand, strict SARSA algorithm can be executed easily by replacing $\max_a Q(s', a)$ with $Q(s', a')$, where a' is an action selected in state s'.

V. Experiments

This section evaluates the proposed method empirically in two learning tasks. Fig.4 shows a slightly modified *rod-in-maze* task in which both the state and the action are three-dimensional. The original rod-in-maze task is proposed

[3]It is supposed that the number of the iteration is enough. Since predicting the lower bounds of this number is a difficult problem, the samples would be likely to approximate (7).

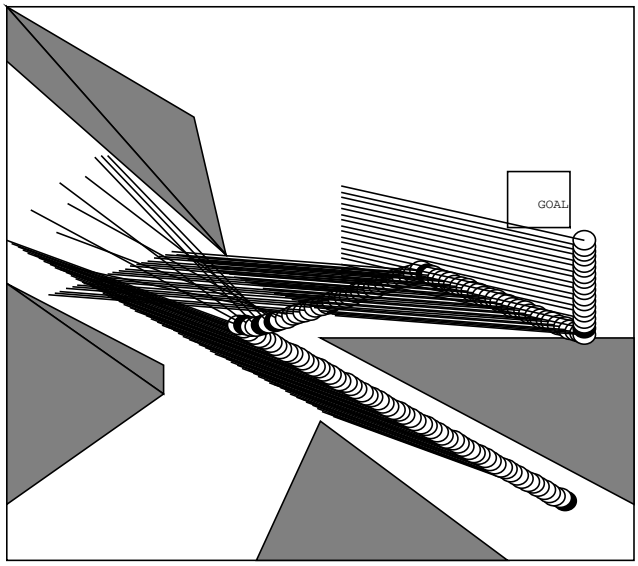

Fig. 4. An example of learned behavior in the rod-in-maze task. The black circles of the bottom of the rod represent the positions at which the learner is making decisions.

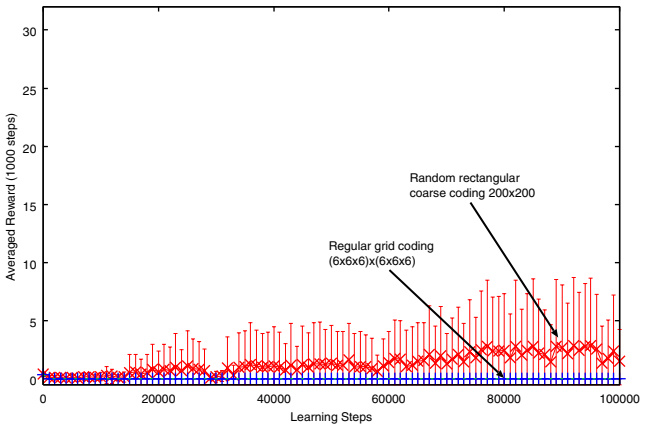

Fig. 5. Learning results averaged over 10 runs in the rod in maze task. The vertical axis gives the averaged reward over 1000 decision steps. The errorbars are standard deviation.

by [6]. The state $s = (x, y, \theta)$ is three-dimensional. (x, y) denotes the position of the center of the rod and must lie in the area $0 \leq x \leq 1$ and $0 \leq y \leq 1$. The θ denotes the angle of the rod from the horizontal, and is constrained to lie in the range $0 \leq \theta \leq \pi$. The action $a = (x_d, y_d, \theta_d)$ is also three-dimensional, representing a destination of the rod in the state. When the destination position is given as the agent's action, the simulation of the movement is executed by summing small displacements $(\delta x, \delta y, \delta\theta)$ in the state . The small displacements are given so that each state variable would reach the destination as fast as possible, but it is constrained such that $\sqrt{\delta x^2 + \delta y^2} \leq \frac{1}{100}$ and $-\frac{\pi}{20} \leq \delta\theta \leq \frac{\pi}{20}$. If the rod collides with obstacles, or reaches the goal region, or reaches the destination, then the agent makes the next decision. The goal region is $0.8 \leq x \leq 0.9$ and $0.6 \leq y \leq 0.7$. The rod has length 0.4. In the field, there exist four polygonal obstacles; Obstacle 1 is a triangle and its apexes are $(1.0, 0.1)$, $(0.5, 0.4)$ and $(1.0, 0.4)$. Obstacle 2 is a triangle and its apexes are $(0.4, 0.0)$, $(0.5, 0.25)$ and $(0.8, 0.0)$. Obstacle 3 is a quadrangle and its apexes are $(0.0, 0.5)$, $(0.25, 0.35)$, $(0.25, 0.3)$ and $(0.0, 0.1)$. Obstacle 4 is a quadrangle and its apexes are $(0.0, 1.0)$, $(0.3, 0.8)$, $(0.35, 0.55)$ and $(0.0, 0.9)$. When the rod reaches the goal region, the agent receives reward 100 and the rod is removed to the initial state. Otherwise, the reward is zero.

In this experiment, all the agents partition the state space into $100^3 = 10^6$ grids, dividing each dimension into a hundred regular partitions. The action space is also partitioned into $100^3 = 10^6$ grids in the same way. Thus, the discrete state-action space grows $100^3 \times 100^3 = 10^{12}$. In our approach, we make 200 randomized rectangular features over the state space. In each feature, the sensitive components are selected from the components of the input vector with the probability 0.6 per component. However, the features that select no sensitive component are thrown away and reproduced. The ranges of the rectangles in the sensitive subspace are randomly selected from the grid (e.g., 100 partitions) by uniform distribution. The features for the action space are generated 200 randomized hyper-rectangles in the same way as the state's. The number of iteration in the Gibbs sampling is 15, the discount factor $\gamma = 0.9$, the temperature keeps $T = 0.4$ constant, and the learning rate $\alpha = 0.2$, $offset(s)$ and $width(s)$ in (7) are given by the average and deviation of Q-values in state s over 20 actions that are selected by uniform distribution. It is compared with a regular grid approach: The state space is uniformly partitioned into $6 \times 6 \times 6 = 216$ regular exclusive hyper-rectangles without gaps, and the action space is also uniformly partitioned into $6 \times 6 \times 6 = 216$ regular exclusive hyper-rectangles without gaps. Note that in the regular grid approaches, the hyper-rectangles correspond to the components of the feature vector and used only for value approximation, therefore the actions are selected from the $12 \times 12 \times 12$ discrete action space in the case of the $6 \times 6 \times 6 = 216$ hyper-rectangles.

Fig.5 compares the performance of the random coarse coding against regular grid approaches averaging over 10 runs. The best solutions of this task are composed of only three decision steps as shown in Fig.4, but unfortunately the proposed method often couldn't find it resulting four or five decisions. The reason of this result is that the random coarse coding generates different rectangles for each trial, therefore the essential features for the optimum solutions couldn't be made in such cases. On the other hand, the uniform grid coding couldn't find any solutions at all even though the number of features is larger than the random coarse coding's one, especially in the case of 6^3 states and 6^3 actions.

As the second example, Fig.8 shows a slightly modified multi-joint arm reaching task in which both the state and the action are 8-dimensional. The original benchmark is also proposed by [6]. The arm is planar and composed of eight links that are connected by the joints in series. The state $\boldsymbol{s} = (\theta_1, \theta_2, \cdots \theta_8)$ is 8-dimensional, θ_1 denotes the angle of the root link from the horizontal, and θ_i where $i > 1$ is

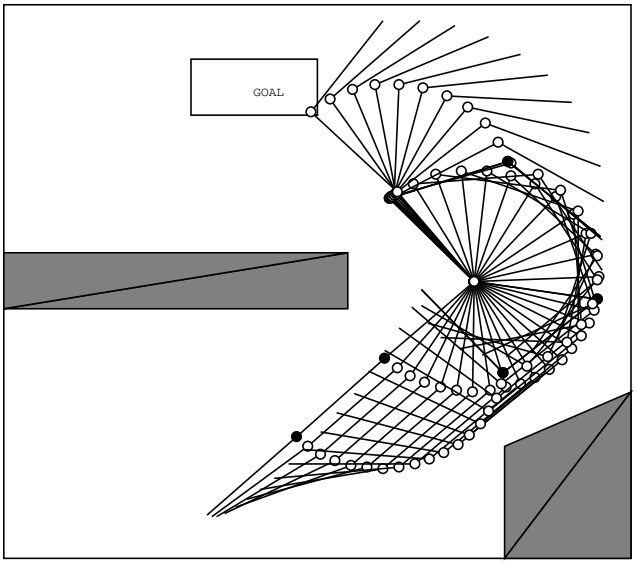

Fig. 6. A learned example behavior of a reaching task using redundant-arm that have three-joints. The conditions that the joints are shown by black circles are the positions at which the learner is making decisions.

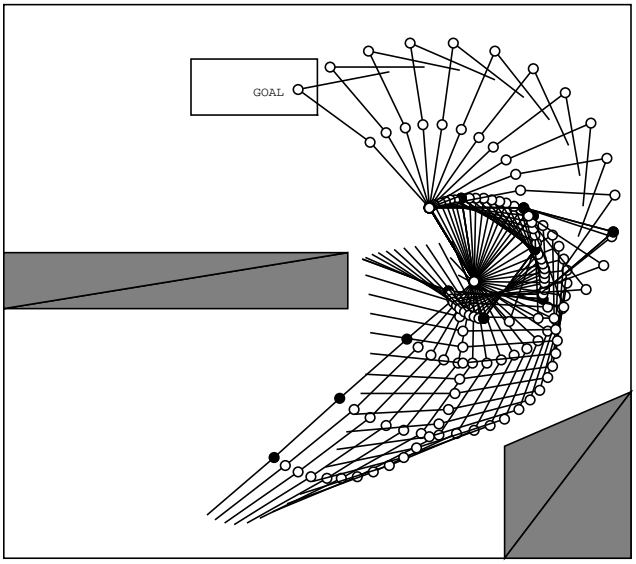

Fig. 7. A learned example behavior of a reaching task using redundant-arm that have four-joints. The conditions that the joints are shown by black circles are the positions at which the learner is making decisions.

the angle of the joint between ith link and $(i-1)$th link. All angles are constrained to lie in the range $-0.75\pi \leq \theta \leq 0.75\pi$. The arm is constrained to lie in the area $0 \leq x \leq 1$ and $0 \leq y \leq 1$. The action $\boldsymbol{a} = (\theta_{1d}, \theta_{2d}, \cdots \theta_{8d})$ is also 8-dimensional, representing a destination of the joint's angles. When a destination is given as the agent's action, the simulation of the movement is executed by summing small displacements $(\delta\theta_1, \delta\theta_2, \delta\theta_8)$ in the state space. It is given by

$$\begin{aligned}(\delta\theta_1, \delta\theta_2, \delta\theta_8) &= \lambda(\boldsymbol{a} - \boldsymbol{s}) \quad \text{, where} \\ \lambda &= \min\left(\frac{\pi}{15|\boldsymbol{a} - \boldsymbol{s}|}\ ,\ 1\right)\end{aligned}$$

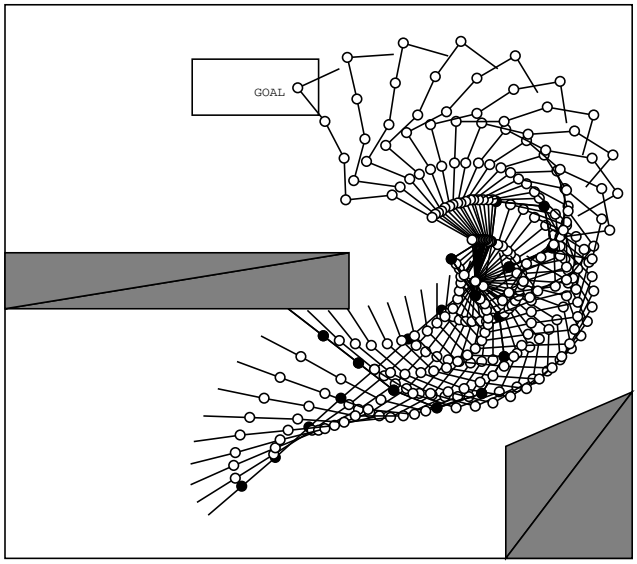

Fig. 8. A learned example behavior of a reaching task using redundant-arm that have eight-joints. The conditions that the joints are shown by black circles are the positions at which the learner is making decisions.

When the arm collides with obstacles, or intersects itself, or reaches the goal region, or reaches the destination, then the agent makes the next decision. The position of the root-joint connected to the base is $(0.75, 0.5)$, and the arm has length 0.6. The goal region is $0.3 \leq x \leq 0.5$ and $0.8 \leq y \leq 0.9$. The initial state is $\theta_1 = -0.75\pi$ and the others are all 0. There exist two polygonal obstacles; Obstacle 1 is a rectangle, and its apexes are $(0.8, 0.0)$, $(0.8, 0.2)$, $(1.0, 0.3)$ and $(1.0, 0.0)$, Obstacle 2 is a quadrangle, and its apexes are $(0.0, 0.45)$, $(0.55, 0.45)$, $(0.55, 0.55)$ and $(0.0, 0.55)$.

In this experiment, all the agents partition the state space into 10^8 grids, dividing each dimension into regular 10 partitions. The action space is also partitioned into 10^8 grids in the same way. Thus, the discrete state-action space grows $10^8 \times 10^8 = 10^{16}$. In our approach, we make 200 randomized rectangular features for the state space and the action space respectively. In each feature, the sensitive components are selected from the components of the input vector with the probability 0.3 per component. The number of iteration in the Gibbs sampling is 40, and the other settings are the same as the rod-in-maze problem's. It is compared with a regular grid approach: the state space is uniformly partitioned into $2^8 = 256$ regular exclusive hyper-rectangles without gaps, and the action space is also uniformly partitioned into $2^8 = 256$ regular exclusive hyper-rectangles without gaps. Note that in the regular grid approaches, the hyper-rectangles correspond to the components of the feature vector and used only for value approximation, therefore the actions are selected from the 10^8 discrete action space.

Fig.11 compares the performance of the random coarse coding against the regular grid approach averaging over 10 runs.

The best solutions of this task are composed of only three decision steps. Although the proposed method generates

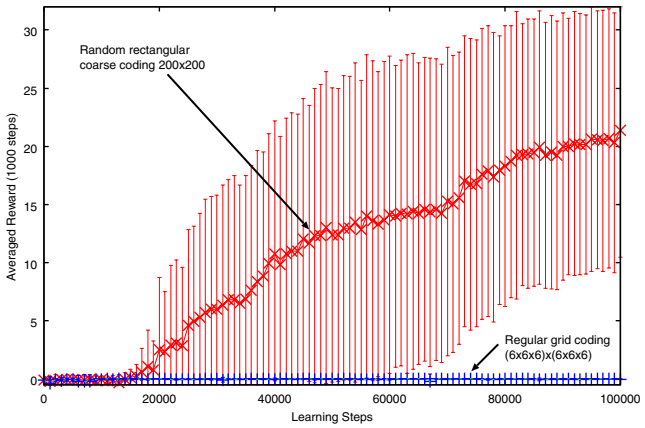

Fig. 9. Learning results averaged over 10 trials in the 3-joints redundant-arm reaching task. The vertical axis gives the averaged reward over making 1000 decision steps.

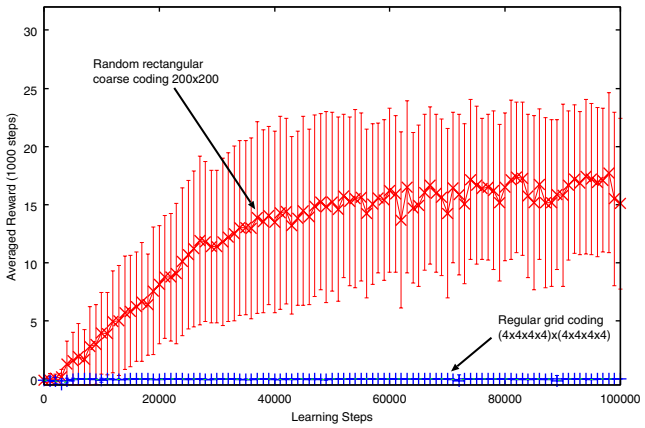

Fig. 10. Learning results averaged over 10 trials in the 4-joints redundant-arm reaching task. The vertical axis gives the averaged reward over making 1000 decision steps.

different features in each trial, it can find the best solution in all the cases. On the other hand, the uniform grid approach couldn't find any solutions even though the number of features are larger than the random coarse coding's.

VI. Discussion

A. Relation between the Pattern of the Random Features and the Learning Performance

In the simulation results, the coarse coding sometimes failed to find optimum solutions because of the randomness of the features. However, it could find approximate solutions using just 200 state features and 200 action features not only in the 3×3-dimensional state-action problem but in the 8×8-dimensional problem as well. Reflection on some of these will make clear that the coarse coding is suited for finding feasible or reasonable solutions in high-dimensional problems that have a number of sub-optimum solutions. That is to say, it is hard to find optimum solutions, especially when the problem has a needle-like evaluation function. The regular grid approaches could not find any solutions at all, though using no less computational resources than the

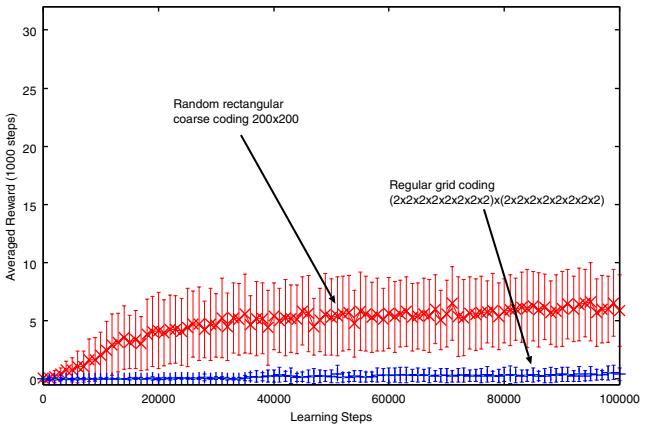

Fig. 11. Learning results averaged over 10 or 13 trials in the 8-joints redundant-arm reaching task. The vertical axis gives the averaged reward over making 1000 decision steps.

coarse coding's. In the grid approaches, each cell exclusively corresponds to a specific binary (0-1-valued) feature, and the set of feature vectors is *always* linearly independent. On the other hand, the coarse coding tends to generate linearly independent feature vectors for essential states (or actions) without exponential growth of the variables, however, the linearly independence between the feature vectors is not guaranteed.

Fig.12 shows landscapes of learned Q-functions in the redundant-arm reaching task. The landscape in the randomized coarse-coding is smooth and better to approximate functions than the regular grid approach.

The learning performance of the coarse coding also depends on the choice of the sensitive components, the ranges and the size of the rectangles for the features. In this paper, these parameters are simply given randomly, but it is also possible to consider some other sophisticated methods; One is *feature iteration approach* [1], whereby a set of features is selected, some computation is done using these features, and based on the results of the computation, some new features are introduced. Indeed, about 30 percent of the coarse-coding features are not used in the rod-in-maze experiments. When we give the coarse coding features randomly, we should fix a few parameters: One is a number of the features, and the other is probability for choosing sensitive components from the input vector to generate rectangular features. In this paper, the probability for sensitive components is experimentally designed so that about 10 percent of the features are active in all the state or action.

This paper introduced a value approximation technique that the state features and the action features are separately given. It is advantageous to execute Gibbs sampling over action space. The other approach can be considered to generate features directly over state-action space, however, it tends to cost heavy computation in Gibbs sampling.

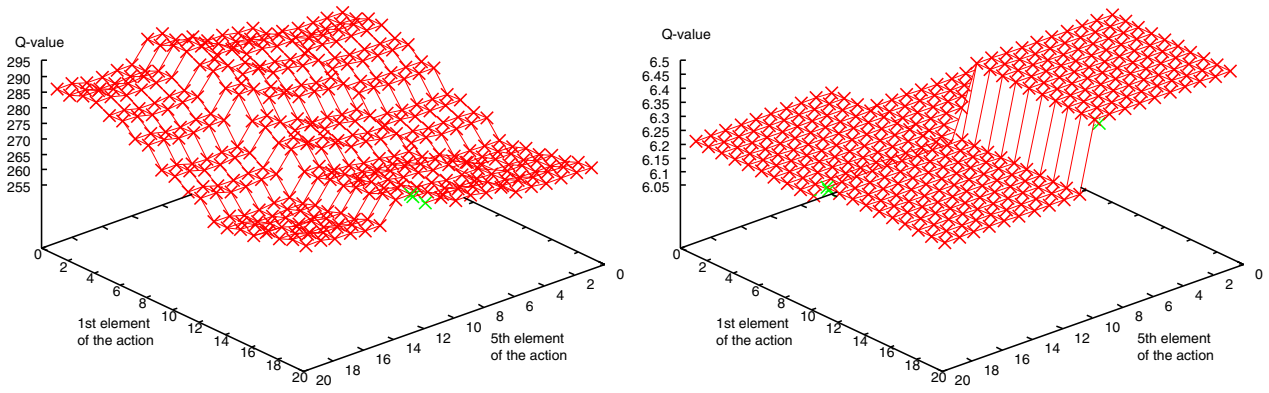

Fig. 12. Lefthand side: A landscape of a learned Q-function using random rectangular coarse coding in the redundant-arm reaching task. The axises are the 1st and 5th element of the action in the initial state. The other action elements are kept constant. Righthand side: A landscape of a learned Q-function using the regular grid approach in the same condition.

B. Finding an Appropriate Number of the Iteration in Gibbs Sampling

In the experiments, the numbers of the iteration are given ad hoc. In general, it is difficult to predict the lower bounds of the number of the iteration in Gibbs sampling. It is noteworthy that the Gibbs sampling affects only the time for action selection, it does not affect the learning steps. Therefore, the number of the iteration should be given as long as possible. To get good samples with less time, overrelaxation techniques or simulated annealing [3] are possible.

VII. Conclusion

This paper proposed a new approach combining random rectangular coarse coding with Gibbs-sampling action selection to cope with reinforcement learning in high-dimensional domains. The proposed coding method is very simple and quite suited both to approximate Q-functions in high-dimensional spaces, and to execute Gibbs sampling. The action selection scheme using Gibbs-sampling enables to handle high-dimensional action space in RL problems. The parameters of the rectangular features are given randomly, however, it works very well in Rod-in-maze problem and Multi-joint arm problem. A future work is to consider some other sophisticated methods to generate (or modify) the feature parameters.

References

[1] Bertsekas, D.P. & Tsitsiklis, J.N.: Neuro-Dynamic Programming, Athena Scientific (1996).

[2] Hengst, B.: Discovering Hierarchy in Reinforcement Learning with HEXQ: Proc.of 19th International Conference on Machine Learning (ICML2002), pp.243–250.

[3] Jordan, M. I.: Learning in Graphical Models, The MIT Press, (1999).

[4] Kaelbling, L. P., & Littman, M. L., & Moore, A. W.: Reinforcement Learning: A Survey, Journal of Artificial Intelligence Research, Vol. 4, pp. 237–277 (1996).

[5] Lin, L. J.: Scaling Up Reinforcement Learning for Robot Control, Proceedings of the 10th International Conference on Machine Learning, pp. 182–189 (1993).

[6] Moore A.W. and Atkeson, C.G.: The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-spaces, Machine Learning, Vol.21, pp.199–233 (1995).

[7] J.D.R. Millan, D.Posenato & E.Dedieu: Continuous-Action Q-Learning, Machine Learning, Vol.49, pp.247–265 (2002).

[8] Sallans, B. & Hinton, G.E.: Reinforcement learning with factored states and actions. Journal of Machine Learning Research, vol.5, pp.1063-1088 (2004).

[9] Santamaria, J. C., Sutton, R. S. & Ram, A.: Experiments with Reinforcement Learning in Problems with Continuous State and Action Spaces, Adaptive Behavior 6 (2), pp.163–218 (1998).

[10] Sutton, R.S.: Generalization in reinforcement learning: Successful examples using sparse coarse coding, Advances in Neural Information Processing Systems 8 (NIPS8), pp.1038–1044 (1996).

[11] Sutton, R.S. & Barto, A.: Reinforcement learning: An introduction, A Bradford Book, The MIT Press (1998).

[12] Takahashi, Y. & Takeda, M. & Asada, M.: Continuous valued Q-learning for vision-guided behavior acquisition. Proceeding of the 1999 IEEE International Conference on Multi-sensor Fusion and Integration for Intelligent Systems, pp.255–260 (1999).

[13] Watkins, C.J.C.H. & Dayan, P.: Technical Note: Q-Learning, Machine Learning, Vol.8, pp.279–292 (1992).