Cooperation of Categorical and Behavioral Learning in a Practical Solution to the Abstraction Problem

Atsushi UENO and Hideaki TAKEDA

Graduate School of Information Science
Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara 630-0101, JAPAN

{ueno, takeda}@is.aist-nara.ac.jp

Received 17 March 1999

Abstract Real robots should be able to adapt autonomously to various environments in order to go on executing their tasks without breaking down. They achieve this by learning how to abstract only useful information from a huge amount of information in the environment while executing their tasks. This paper proposes a new architecture which performs categorical learning and behavioral learning in parallel with task execution. We call the architecture *Situation Transition Network System (STNS)*. In categorical learning, it makes a flexible state representation and modifies it according to the results of behaviors. Behavioral learning is reinforcement learning on the state representation. Simulation results have shown that this architecture is able to learn efficiently and adapt to unexpected changes of the environment autonomously.

Keywords Abstraction Problem, Categorical Learning, State Space Segmentation, Reinforcement Learning, Interleave Planning

§1 Introduction

Recent developments in robotic technologies have diversified the roles of robot to cater to different fields of requirements. At one end, the rover *Sojourner* has played an active part in the *Mars Pathfinder* mission operated by NASA and

at the other end, there are home robots such as pet robots, welfare robots and housekeeping robots which have been or will soon be available in the market. These robots are expected to cope with various tasks in various environments. However, these robots face common difficulty in executing their tasks, mostly due to the ever-changing environment. The fact that the real world is a dynamic environment and has a huge amount of information, requires autonomous agents to cope with and process these masses of information in a limited frame of time in order to execute their tasks in a smooth manner. Many problems occur, such as explosion of processing time, explosion of the amount of description, and inconsistency between the real world and the internal model if they cannot fulfill this requirement. To overcome this problem, agents must be able to abstract only necessary information relevant to their tasks. This problem is called *the abstraction problem*.

For artificial agents, one of the most common solutions to this problem is "designing how to perform abstraction in advance" and this is commonly adopted by industrial robots and vending machines. While this method is very efficient when the agents are in a familiar environment or an environment that the designers have already foreseen, it is not as reliable when the agents need to take appropriate actions in a environment that the designers have not taken into account. Therefore, to move and function in various environments, agents should possess the ability to "learn how to perform abstraction empirically" in order to get acclimated to the unfamiliar environments gradually. This can be regarded as categorical learning because abstracting common properties from concrete things is regarded as putting them into the same category.

In this paper, we discuss how an agent constructs and maintains categories for recognizing its environment and solve the abstraction problem practically. We propose a new architecture which enables robots to learn a flexible internal representation while executing the task. We call the architecture *Situation Transition Network System (STNS)*. This is a solution to the problem how to adapt autonomously to various environments and their changes. This architecture allows a robot to abstract only relevant information from its environment for the execution of its current task.

The remainder of this paper is structured as follows. In Section 2, we discuss categorical learning in a robot. In Section 3, we clarify assumptions on problems to which STNS can be applicable. Then, we give an outline of STNS in Section 4. In Section 5 and Section 6, we explain categorical learning and

behavioral learning in STNS respectively. In Section 7, we show the validity of STNS in the experiments of computer simulations. Finally, we conclude the paper in Section 8 by showing the advantages of our architecture and discussing future work.

§2 Categorical Learning in a Robot

2.1 Categorical Learning for Robots

Categorical learning (or, category learning[5]) is usually used for reproducing human recognition on a computer. In this case, human beings decide the meanings of categories (in other words, what kind of information to recognize). On the other hand, in case of categorical learning in a robot, meanings of categories can be decided on the criterion how well the robot can act. In this sense, this type of learning is more autonomous than usual pattern recognition learning and usual concept learning.[3]

In robot learning, human designers usually fix categories for recognition, and behavioral learning is performed on internal representation formed from the categories. However, human-designed categories are not always suitable for the robots. By letting categorical learning and behavioral learning cooperate, it is expected that a robot can have flexibility for adapting to unfamiliar environments. Such dual learning is realized only in the domain of reinforcement learning.

2.2 Categorical Learning for Reinforcement Learning

Reinforcement learning[15] is useful for robots to acquire appropriate behavior because it can work with a small amount of a priori knowledge. In general, reinforcement learning is regarded as learning of a behavior policy that maximizes discounted sum of the rewards received over time. The sum is called the utility of the state (see subsection 6.1). For this purpose, many reinforcement learning methods perform the approximation of the utility function (or the action-value function).

In the real world, almost all robots have real-valued sensors and a continuous input space (state space). There are two approaches for approximating the utility function in the continuous state space: using an implicit representation of the function such as a neural network or segmenting the continuous space into discrete states in which the value of the utility function is regarded

as invariable.

As the former, several kinds of implicit representations can be used such as perceptron-like neural networks,[2, 17, 20, 9, 10] CMAC (Cerebellar Model Arithmetic Computer),[18] and RBF (Radial Basis Function).[4] But this approach has three problems: first, it is difficult to understand the internal representation. Second, therefore, it is very difficult to analyze the convergence and the optimality of learning. Third, therefore, the human designer must fix many parameters arbitrarily on the basis of her/his own experience. It is difficult for her/him when the task of the robot is complex.

As the latter, the most usual representation is the grid representation which the human designer must make by dividing the continuous space at equal or arbitrary intervals. It is also difficult for him in case of a complex task. If the representation is too coarse, there occurs so-called *perceptual aliasing problem*, i.e., the robot cannot discriminate the important states for the task. On the other hand, too fine representation leads the robot to explosion of learning time.

In order to cope with this difficulty, many methods have been proposed which can segment the state space autonomously. This segmentation can be regarded as a kind of categorical learning. These methods can be divided into six classes by the policies of segmentation: methods in the first class segment the state space enclosing input vectors from which the robot reaches the same reward by the same sequence of behaviors in the same state (*segmentation based on behaviors*).[1, 6, 22] Methods in the second class segment the vicinities of experienced rewards.[8, 21] Methods in the third class segment the state space finer in the areas which have higher density of input vectors.[7, 12] Methods in the fourth class segment the state space by clustering of input vectors.[14] Methods in the fifth class segment the state space enclosing input vectors which have linear relation with their gradients by the same behavior.[19] Methods in the sixth class divide states from which the robot cannot predict state transition.[13, 16]

In this paper, we focus on segmentation based on behaviors. The reason is as follows: reinforcement learning on discrete state representation assigns one feasible behavior to each state. Therefore, necessary and sufficient state representation for the task is realized by putting input vectors at which the robot should take the same behavior for achieving the task together into the same state. By segmentation based on behaviors, input vectors at which the robot should take the same behavior for receiving the same reward put together into the same state. Thus, this policy makes highly abstracted state representation

specialized to the current task. In this representation, each state is regarded as a path to a reward.

There are three previous works on this policy. Asada et al.[1] proposed a method that divides the state space by hyper-ellipsoids which enclose input vectors from which the robot achieves the goal or already acquired state by a variable sequence of one kind action primitive. Yairi et al.[22] proposed a method that segment the state space using Bayesian classifier on the basis of the same segmentation policy. Ishiguro et al.[6] proposed a method that divides states by hyper-planes in which the robot receives different rewards (or delayed rewards) when executing the same action. These methods perform categorical learning in an off-line way: the system segments the state space on the basis of experiences of random (or given) behaviors in learning phase, then executes the task with the fixed state representation. If the representation is unsatisfactory, these methods can add new states (or boundaries) by off-line learning but cannot modify or eliminate existing states (or boundaries).

In this type of categorical learning, a main problem is insufficiency of adaptability to environments. In order to solve this problem, we think that it is necessary to learn the state representation in on-line way: the system segments the state space while executing the task on the basis of experiences of task-executing behaviors. We aim to realize on-line learning system with the following five functions:

1. Abstracting a state aggressively from a few data.
2. Modifying state shapes gradually reflecting new experiences.
3. Maintaining an appropriate state representation stably against contingent deviation of the distribution of data.
4. Eliminating states which have already been useless.
5. Representing finer shapes of states incrementally as data increase.

This type of on-line learning contributes to the following three advantages:

autonomy The system has no phases such as learning phase or task-executing phase. Therefore, human users need not worry when they should switch phases.

adaptability The system can adapt flexibly to small changes while keeping high performance. The learning system which fixate the state representation must learn again from the beginning in order to adapt even to small changes.

efficiency of learning By early abstraction of states, the flow of behaviors through the states to rewards is formed, and many data useful for representing states are collected efficiently. Therefore, the system has good performance even in the early stage of learning, and learning converges quickly.

On the other hand, the extra processing time for maintaining states is a disadvantage of this type of on-line learning. However, it does not matter in the real robot because it is far shorter than time for moving the robot. Another disadvantage is that this type of learning has difficulty in constructing an elaborate state representation because the representation always fluctuates slightly.[*1] This represents the difference of purposes between off-line learning and on-line learning. Off-line learning is regarded as a kind of empirical design of state representation. The purpose is constructing an elaborate state representation from a large quantity of data. On the other hand, on-line learning is regarded as a function of adapting the state representation continuously to changes of the environment. The purpose is continuing task execution without breaking down.

Our developed architecture, STNS, performs segmentation based on behaviors by on-line learning. In order to realize the above five functions of the aimed on-line learning system, we developed a new flexible state representation, the *bitten hyper-ellipsoid representation* (see subsection 5.2), and five autonomous operations for maintaining the state representation (see subsection 5.3). In this paper, we call the highly abstracted flexible state by this representation a *situation*.

§3 Assumptions

Our method can be applied to the problems which satisfy the following conditions:

1. The state space is continuous and multidimensional.
2. The task is specified by rewards, but it is restricted to looking for big rewards. STNS cannot be used for escaping from big danger.[*2]
3. Output is selected out of several behaviors, not in a continuous action space. A variety of behaviors is admitted, i.e., from a simple action

[*1] Of course, the fluctuation should be decreased. This is mentioned as the above third function of the aimed on-line learning system.

[*2] The reason is that situations construct paths to big rewards in segmentation based on behaviors (see subsection 2.2).

such as move forward and so on, to a complex behavior such as wall-following and so on.

4. The problems are solved by finding a feasible behavior policy, not necessarily the optimum one. STNS aims for the optimization in the sense of minimizing the number of behaviors, not in the sense of minimizing consumed time, energy, or cost.
5. The system dynamics and control laws are unknown.
6. The Markov property holds: the transition probabilities from any given state depend only on th state and not on previous history.[15]

These conditions are used in common in the studies of segmentation based on behaviors.[1, 6, 22]

§4 Situation Transition Network System

As shown in Fig. 1, STNS consists of a situation classifier, a situation transition network (STN), and several behavior modules. In each behavior step, the system identifies the current situation where the current input should be included, then makes a partial plan on the STN, and at last activates a behavior module according to the plan.

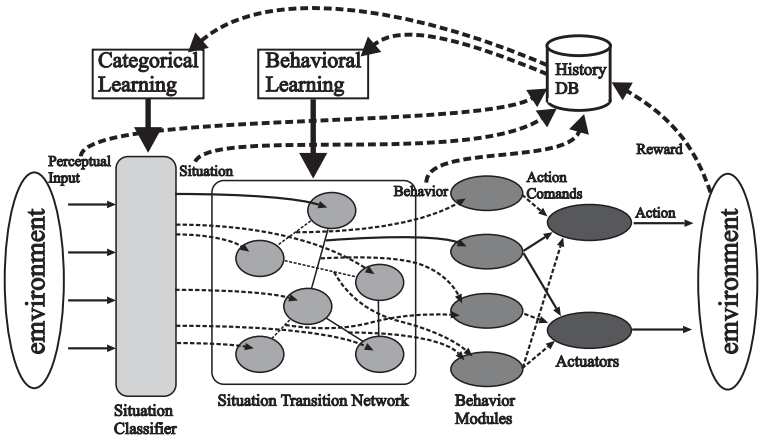


Fig. 1 The structure of STNS

Categorical learning and behavioral learning are performed on the data which consists of the input, the corresponding situation, the selected behavior, and the received reward. This data puts into a history database. It keeps the data for a fixed period and deletes it after that. Therefore, STNS can adapt

flexibly to the changes of environment and the internal representation without being misguided by outdated data.

In categorical learning, the system performs segmentation based on behaviors by on-line learning: it extracts and maintains situations in the continuous state space on the basis of experiences of task-executing behaviors. Reinforcement learning of a behavior policy is performed on this situation representation: the system constructs an MDP (Markov Decision Problem[*3]) model of the environment which consists of the transition probabilities between situations and the expectations of immediate rewards accompanying transitions by the maximum likelihood estimation. It decides an appropriate behavior by partial planning on the model in order to maximize discounted sum of rewards received over a period of time. In each behavior step, the system adjusts the shapes of situations in categorical learning and modifies the MDP model in behavioral learning reflecting the last data. If big modification is needed, the system extracts or eliminates situations in categorical learning and modifies the MDP model reflecting the new situation representation in behavioral learning on the data in the history database. In this way, these two learning processes are performed in parallel while executing the task.

In the following two sections, we explain the categorical learning and the behavioral learning respectively in more detail.

§5 Categorical Learning in STNS

5.1 Segmentation Based on Behaviors in STNS

STNS segments the state space into some situations. In STNS, a situation is defined as a set of input vectors from which *the system can meet with the specific result by the specific behavior*. The specific behavior is called the *condition behavior* of the situation. The specific results are divided into two types, i.e., *R-situation* and *T-situation*. In a situation based on immediate rewards called R-Situation, the result is to receive a specific big reward (called the *goal reward*). In a situation based on situation transitions called T-Situation, the result is to transit to a specific situation (called the *parent situation*). If every chain of T-Situations is anchored to an R-Situation, every situation is guaranteed to lead to a goal reward by the same sequence of behaviors. Thus,

[*3] MDP is the problem of calculating an optimal policy in an accessible, stochastic environment with a known transition model.[15]

a segmentation based on behaviors which encloses input vectors from which the robot reaches the same reward by the same sequence of behaviors in the same situation is realized.

Every situation has two attributes: the condition behavior and the specific result (the goal reward or the parent situation). Values of these attributes are decided automatically by categorical learning (see subsection 5.3).

5.2 Bitten Hyper-Ellipsoid Representation

In order to realize the first, the second, the third, and the fifth function of the aimed on-line learning system mentioned in subsection 2.2, we propose the bitten hyper-ellipsoid representation. In this representation, each situation is shaped by the positive instances and the negative instances that are decided based on the definition of the situation. A positive instance of a situation is an input vector in the history database from which the system met with its specific result by its condition behavior. A negative instance of a situation is an input vector classified into the situation from which the system did not meet with its specific result by its condition behavior.

As shown in Fig. 2, this representation is a mixture of macroscopic recognition and microscopic recognition. In the macroscopic recognition, the boundary of a situation is a contour of Mahalanobis' distance from the population of the positive instances.[*4] This boundary forms a hyper-ellipsoid. The microscopic recognition is realized by Nearest Neighbor methods.[*5] Nearest Neighbor methods are popular especially in the domain of pattern recognition.[11)] The distance used in the microscopic recognition is not Euclidean distance which is usually used in Nearest Neighbor methods, but the distance which is standardized by the standard deviation of the population of the positive instances. When the system discriminates whether the current input vector belongs to each situation or not, first, the system checks whether it is included in the hyper-ellipsoid or not (macroscopic recognition). Then, if it is included, the system discriminates whether it belongs to the situation by Nearest Neighbor methods (microscopic recognition).

In the macroscopic recognition, the processing time for discrimination and learning is very short because a hyper-ellipsoid is represented by some simple parameters (the sample mean and the sample covariance matrix of the positive

[*4] [1)] uses the same type of representation.

[*5] [19)] uses the same type of representation.

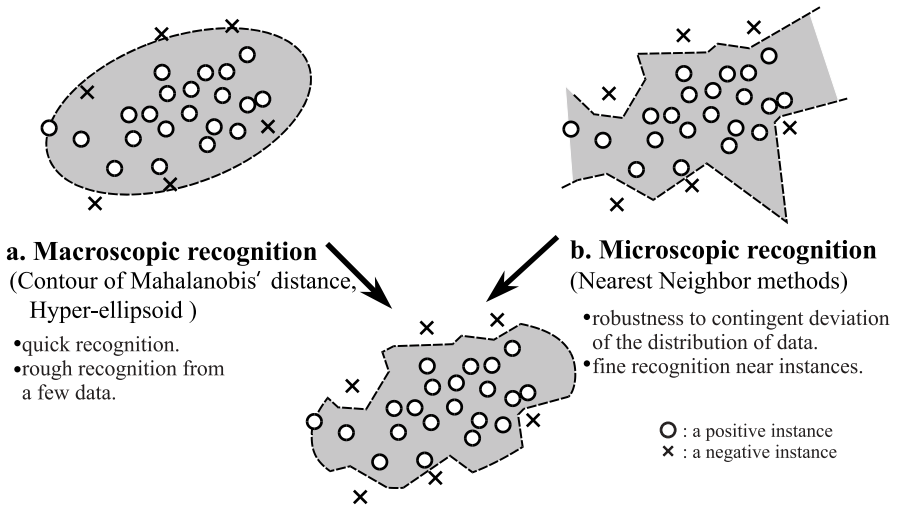


Fig. 2 The bitten hyper-ellipsoid representation

instances). Furthermore, it can draw a rough shape of a situation from a few data. On the other hand, it is inevitable to classify data into an incorrect situation even if there are a great number of data when the distribution of data is not normal distribution.

The microscopic recognition can discriminate very precisely in the vicinity of each instance. Therefore, it grows to be able to discriminate more precisely all over the space as data increase. On the other hand, it cannot discriminate appropriately at a distance from instances. Therefore, it cannot represent an enclosed shape from a few data. Even after learning has progressed, the negative instances are apt to be short because collecting many negative instances is in conflict with achieving high performance in task execution. Therefore, several rifts are formed as shown in Fig. 2b. These rifts are covered by combining with macroscopic recognition as shown in Fig. 2c. In this recognition, the processing time for learning is very short because all of the process is only memorizing the position of the instance. On the other hand, it needs considerably long processing time for discrimination and much memory space. STNS can save the processing time by eliminating candidates in the macroscopic recognition before this time-consuming process.

In regard to the second function in subsection 2.2, addition of new data

and subtraction of old data are easily performed on both representation for the macroscopic recognition and the microscopic recognition because the former consists of the sample mean and the sample covariance matrix of the positive instances and the latter consists of positive and negative instances themselves. In regard to the third function, the macroscopic recognition is fragile to deviation of the distribution of data because the hyper-ellipsoid is dragged to the sample mean of the positive instances. However, the microscopic recognition is not affected by the deviation if there are plenty of data near the boundary. Thus, each situation can maintain a stable shape as a whole. In regard to the first function, STNS can represent a rough shape of a situation from a few data mainly by the macroscopic recognition. In regard to the fifth function, STNS can represent finer shapes of situations incrementally as data increase by using the microscopic recognition in the critical areas.

5.3 Categorical Learning in STNS

In STNS, the state space is divided into overlapped bitten hyper-ellipsoids (situation 1–7) and a margin space (called *situation 0*, hereafter) as shown in Fig. 3a. Each different shaded area denotes a situation. A new perceptual input is ascertained whether it belongs to each situation from the top of the discrimination tree in Fig. 3b.

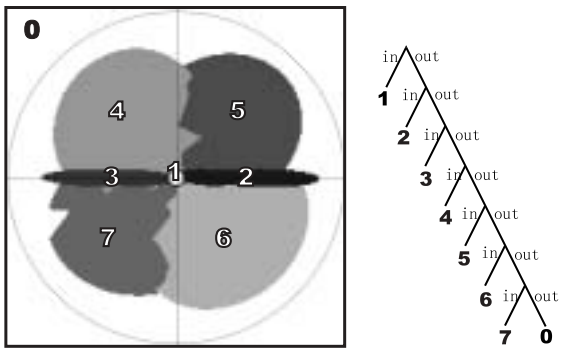


Fig. 3 The state space and the discrimination tree

Categorical learning is realized by extraction and maintenance of situations. The learning starts from the empty state space (that is, only situation 0), and progress while segmenting the state space. For example, first, R-Situation

A is extracted from which the robot can reach a certain big reward by a certain behavior, and then T-Situation B is extracted from which the robot can reach R-Situation A by a certain behavior, ..., and so on. The shape of each situation is not fixed but maintained to adjust to the current environment.

There are following two conditions for situation extraction:

1. **R-Situation** There are enough (more than N^R_{init}) data in the history database each of which shows that the system received a specific reward larger than a threshold (r^R_{min}) by a specific behavior.

   T-Situation There are enough (more than N^T_{init}) data in the history database each of which shows that the system transited from situation 0 to another specific situation by a specific behavior and did not receive a large (more than r^R_{min}) reward.
2. There exists no situation whose attribute values are equal to the specific behavior and the specific result in condition 1.

When both of these conditions are fulfilled, a new situation is extracted. In extraction, the new situation is defined by the specific behavior and the specific result in condition 1, and positive instances are collected from the history database. Negative instances do not exist at the beginning because they are collected among data which are classified into the situation.

In order to realize the second, the third, and the fourth function of the aimed on-line learning system mentioned in subsection 2.2, STNS has the following five autonomous operations for maintaining the situation representation:

1. Updating the populations of the positive and negative instances of situations in each behavior step.
2. Adjusting the boundary of each hyper-ellipsoid so that it encloses just all the positive instances.
3. Rearranging situations in the discrimination tree in order of closeness to the goal rewards.
4. Checking the validity of each situation, and redefining or eliminating inappropriate situations.
5. Eliminating useless situations.

In regard to the second function, the first and the second operations update the shapes of situations gradually reflecting the result of the behaviors of the robot.

In the first operation, the system includes the latest input vector among the positive instances of situations whose condition behaviors are the same with the latest behavior and whose specific results have been realized. Furthermore, it includes the vector among the negative instances of the situation into which the vector was classified if its condition behavior is the same with the latest behavior and its specific result has not been realized. Then, the system eliminates the oldest data from the history database and all populations of the positive and negative instances. This operation is executed in every behavior step. In the second operation, the system checks Mahalanobis' distance from the population of the positive instances to every positive instance at regular intervals and settles the boundary of every hyper-ellipsoid just on the farthest positive instance.

The third operation is effective to the third function. In segmentation based on behaviors, several situations with different attribute values have potentiality of occupying an area. The closest situation to the goal rewards is most valid among them. STNS does not always extract situations in order of closeness to the goal rewards because it extracts them aggressively as mentioned in the first function.[*6] Therefore, the third operation rearranges situations in order of closeness to the goal rewards for the purpose of maintaining an appropriate situation representation. This operation contributes also to the stability of the representation because a situation near a goal reward collects many instances on task execution and therefore its shape is precise and stable. In this operation, the system calculates the utilities of all situations at regular intervals and rearranges situations in descending order of utility.

The fourth operation is also effective to the third function. An "inappropriate situation" in the above description of the fourth operation is a situation whose attribute values are different from the optimum ones found by behavioral learning.[*7] STNS does not always assign the optimum values when extracting a situation because of the aggressive way of extraction. Furthermore, the optimum values change occasionally owing to changes of the environment or the robot itself. Situations become inappropriate for these causes. This operation recycles inappropriate situations if possible, and eliminates them in the other cases. It contributes also to the stability of the situation representation because recycling does not change the representation instantly unlike eliminating. In this

[*6] A T-Situation cannot be extracted earlier than its parent R-Situation. However, it can be extracted earlier than another R-Situation.

[*7] If the MDP model reflects the environment well, the attribute values found by behavioral learning are the optimum in the area of the situation.

operation, the system checks the validity of each situation in each behavior step, and judges the following situations inappropriate:

- the T-Situation whose attribute values are different from the majority of pairs of *the selected behavior in the situation* and *the reached situation* in the history database
- the situation which has too few positive instances to maintain (less than N^R_{min} in case of R-Situation, and N^T_{min} in case of T-Situation)
- the T-Situation whose parent situation was eliminated

If there are inappropriate situations, the system checks whether each of them can be recycled. In checking, the most frequent behavior taken from the situation in the history database is selected as the new condition behavior, and the situation reached most frequently by the behavior is selected as the new parent situation. Then, in the following five cases, the situation under reconstruction is eliminated because the new attribute values are inappropriate:

- The parent situation is situation 0.
- The parent situation is itself.
- The parent situation is a T-Situation whose parent situation is the situation under reconstruction.
- There is already a situation with the same attribute values.
- There are too few (less than N^T_{init}) positive instances decided by the new attribute values in the history database.

In the other cases, the attribute values of the situation under reconstruction are changed to the new ones.

In regard to the fourth function, the system eliminates the situation which has no appropriate attribute values in the fourth operation and the situation with negative utility in the fifth operation.

§6 Behavioral Learning in STNS

In STNS, each situation has a condition behavior. It, however, decides a behavior policy by reinforcement learning. The main reason is that the categorical learning in STNS is on-line learning: it extracts a situation from a few data and it is assumed that the environment can be changed, therefore the initial condition behavior is not always the optimal behavior.

In this section, we propose a new reinforcement learning method, *Interleave Planning-based Reinforcement Learning (IPRL)*, which is suitable for STNS which has a flexible situation representation.

6.1 STN

An STN is an MDP model which consists of the transition probabilities between situations and the expectations of immediate rewards accompanying transitions. The transition probability from situation i to situation j by behavior b is called $p(i, j; b)$, and the expectation of the immediate reward which is received by the transition is called $r(i, j; b)$.

Since the Markov property holds, the problem of finding the optimal policy on an STN is a Markov decision problem. Accordingly, the optimal policy is found by solving the optimal equation of dynamic programming:

$$U(i) = \max_{b \in B} \sum_{j \in X} p(i, j; b)\{r(i, j; b) + \gamma U(j)\} \quad (1)$$

where X is the state space, B is the behavior space, i ($i \in X$) is a situation, and γ ($0 \leq \gamma < 1$) is a discounting factor. $U(i)$ is the utility of situation i, which is the expectation of the discounted sum of the rewards received over time by the optimal policy.

In STNS, the MDP model cannot be given in advance since the situation representation is changed dynamically. Therefore, the model must be estimated at the same time as the learning of a behavior policy. This problem can be solved by reinforcement learning.

6.2 IPRL

STNS uses a new reinforcement learning method, IPRL, for behavioral learning. Many reinforcement learning methods keep the estimated utilities of situations, and improve them gradually after every behavior to converge them to the optimal values. On the other hand, IPRL does not keep them but calculate them when necessary. Therefore, it can adapt to the changes of situation representation immediately. For the calculation, IPRL uses an interleave planning method. Planning in IPRL makes a desirable behavior sequence by forward search from the current situation on the STN.

Interleave planning performs partial planning and plan execution alternately. As a result, it can balance between reactiveness and deliberativeness. Yamada[23] proposed interleave planning which determines the timing to switch planning into execution by the success probability of a plan. STNS uses a method similar to this, which limits the search space by the success probability. By this method, a short plan is made and executed reactively in unfamiliar environments, and a long plan is made and executed deliberatively in familiar environments.

Therefore, it can balance between reactiveness and deliberativeness automatically.

A plan P in STNS is a list of pairs of a behavior and the *target situation* of the behavior:

$$P = ((b_1, d_1), (b_2, d_2), ..., (b_n, d_n))$$

where n is the length of the plan, b_i is the ith behavior in the plan, and d_i is the target situation of the behavior. The success probability of a plan is defined as the product of all transition probabilities in the plan.

IPRL calculates approximate utilities of situations using the optimal equation (Equation 1). The transition probability $p(i, j; b)$ and the expectation of the immediate reward $r(i, j; b)$ are estimated by the maximum likelihood estimation from the data in the history database.

The algorithm of IPRL is as follows:

1. Plans are developed forward from the current situation within the limits that the success probability is larger than p_{min} and the length is shorter than n_{max}.
2. The utilities of all situations which correspond to leaves of the search tree are assumed 0.
3. The planning backtracks from leaves to the root (the current situation). In every node, the utility of the situation is calculated by Equation 1. And a pair of the behavior which maximizes the right side of the equation and situation j which maximizes the inside of $\sum$ is fed back to the parent node as an element of a plan.
4. When backtracking returns to the root, only one plan remains. This is the picked plan, and called the *master plan*.

Then, the system starts to execute the master plan greedily in the environment. Plan execution is stopped and planning is started again after the last behavior in the master plan is executed, or another situation than the target situation is reached.

In IPRL, the optimality of the selected behavior is not guaranteed because it is a kind of approximation. On the other hand, it has the advantage that it can balance between reactiveness and deliberativeness automatically as mentioned above. Furthermore, both process and convergence of the learning are very quick because IPRL performs only the maximum likelihood estimation of the MDP model as learning. This quick convergence of learning is important

to STNS because the internal state representation changes flexibly. The processing time for planning increases exponentially as the depth of planning increases. However, IPRL can avoid an explosion of planning time because it limits the search space by the success probability and the length of a plan. Furthermore, STNS makes a highly abstracted situation representation, therefore, planning toward a big reward is expected not to be very deep. In these points, IPRL is a suitable reinforcement learning method for a component of STNS.

The policy iteration algorithm (PIA)[15] of dynamic programming is another promising method for behavioral learning in STNS. It is a well known method for calculate utilities and the optimal policy on an MDP model. For small state spaces, it is often the most efficient approach.

The processing time of IPRL is dependent on the depth of planning toward a reward. On the other hand, the processing time of PIA is dependent on the number of situations. Therefore, in the environment where many different rewards are received in different places, IPRL may be superior to PIA.

Another merit of IPRL is that the system works with a plan. Because of this merit, the system can explain its behavior. Furthermore, the target situation in the plan can be used as a stopping condition of a continuous behavior. As shown in Fig. 3, the situations made by STNS has rugged shapes. Therefore, this sort of stopping condition is useful in order not to be caught in a projection of an undesirable situation.

§7 Experiments

7.1 Experiment 1: Navigation on 2-D Input

We tackled a simple navigation problem on 2-dimensional input as the first application of STNS. Fig. 4 shows the problem.

We designed STNS for learning a behavior policy of this problem, and conducted an experiment on computer simulation. We set parameters as follows: $N^R_{init} = N^R_{min} = 3, N^T_{init} = 15, N^T_{min} = 7, r^R_{min} = 5.0, \gamma = 0.7, p_{min} = 0.1$, $n_{max} = 7$. Note that the number of positive instances for R-Situation extraction, $N^R_{init} = 3$, is the minimum for constructing an ellipsoid in the 2-dimensional state space. This early extraction accelerates learning (see subsection 2.2). The minimum number of positive instances for R-Situation maintenance, $N^R_{min} = 3$, is also the minimum for construction an ellipsoid. r^R_{min} is a parameter for distinguishing big worthy rewards from trivial ones, therefore, it is set somewhat

Task:
To navigate a 16x12 rectangle rover from an arbitrary position to the only goal (a small circle whose radius is 5) in a 100x100 square continuous plane.
The goal is settled at an arbitrary position in the 72x72 square area at the center of the room.
There is no obstacle but the wall around the room.

Perceptual Input (2-dimensinal):
The real-valued (x, y) coordinates of the goal on the coordinate system fixed to the rover.
The origin is settled at the center of the rover, and the x-axis points to the front of the rover.

Rewards:
1. Arrival at the goal: +10
2. Collision with the wall: -1
3. Trying to rotate over 90 degrees: -1
(The rover can look any direction by at most 90 degrees rotation because of the symmetry of actions.)

Behavior:
a sequence of the same simple actions which is continued until one of the following stopping conditions is fulfilled.
Actions: forward movement, backward movement, clockwise rotation, counterclockwise rotation.
(The rover is assumed to be able to make collision-free rotation.)

Stopping Conditions of Behaviors:
1. Getting some reward.
2. Arrival at the current target situation in the master plan.

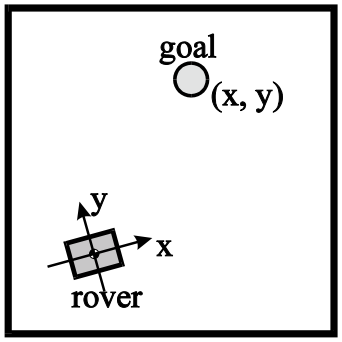


Fig. 4 The navigation problem on 2-D input

lower than the reward for arrival at the goal, 10. The other parameters are set empirically. If N_{init}^T is too small, many needless and obstructive situations may be constructed. Inversely, if it is too large, learning converges slowly. If N_{min}^T is too small, deformed and obstructive situations are not readily eliminated and this may prevent the system from adapting flexibly to changes. Inversely, if it is too large, necessary situations may be eliminated unluckily, therefore the performance may become unstable. As the discounting factor, γ, increases toward one, future rewards become more significant and learning becomes more exploitation-oriented. Inversely, as it decreases toward zero, learning becomes more exploration-oriented. p_{min} and n_{max} are concerned with deliberativeness of planning. If p_{min} is small and n_{max} is large, planning is deliberative, and learning may be stable, but time for planning increases. The number of data in the history database is 1000. The larger it becomes, the more precise the situation representation may becomes, but the less flexibly the system adapts to changes. Both the boundaries of all hyper-ellipsoids and the order of situations are adjusted every 100 behavior steps. The shorter the interval becomes, the more precisely the system reflects its experiences, but the more time this process consumes. The number of behavior steps is the sum of the number of behaviors

and the number of arrivals at the goal. It is equal to the number of data which the system collected by the time.[*8]

Every trial starts after the goal is settled at an arbitrary position and the rover is set at an arbitrary position outside of the goal in an arbitrary direction. The trial ends when the rover arrives at the goal, and the next trial starts immediately. A trial set is a set of consecutive trials in which learning is continued. At the beginning of each trial set, there are only the goal area and situation 0 (the margin space) in the state space. We conducted 20 trial sets.

Fig. 5 shows a typical state space after learning has converged and the optimum state space. Each different shaded area denotes a situation and the arrow in it denotes the condition behavior of the situation. The small circle (the radius is 5.0) at the center of each space denotes states in which the rover arrives at the goal. The big circle (the radius is about 112) surrounding each space denotes the longest distance between the rover and the goal. Input vectors on this circle are received when the rover and the goal are located accidentally at opposite corners of the room. Therefore, the density of data is very low near the circle and these areas remain as the margin space. As shown in this figure, a good situation representation and a good behavior policy were acquired. In every trial set, a similar good representation were acquired. The goodness of the situation representation is proved by the high success probabilities of behaviors. After convergence (9001–10000 behavior steps), the average success probabilities of behaviors from R-Situations and T-Situations were 94.3% and 92.1%, respectively.

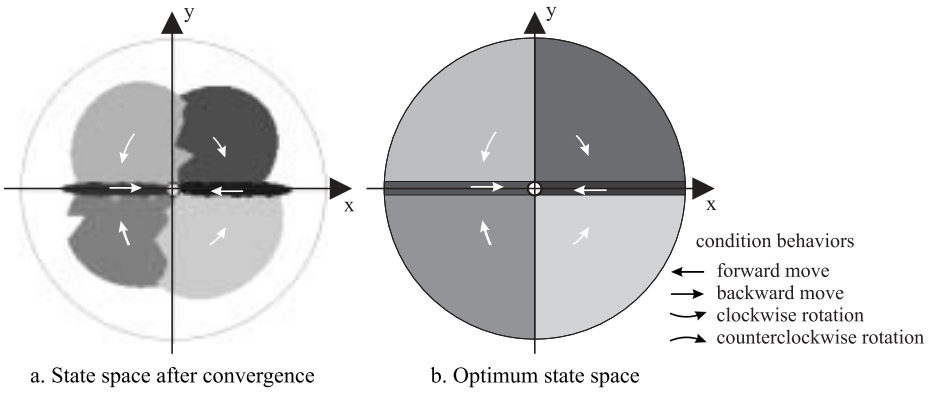


Fig. 5 A state space after convergence and the optimum state space

[*8] When the rover arrives at the goal, the system picks a data but does not take a behavior.

Fig. 6 shows the average number of behaviors which are needed to reach the goal. All trial sets were partitioned into blocks of 100 behavior steps, and the number of behaviors was averaged in each block. Learning has converged after about 1200 behavior steps on average. In this experiment, the rover has no heuristics for acquiring big rewards and selects behaviors at random in the early stage of learning. Therefore, it can arrive at the goal only 2.04 times in every 100 behavior steps. The number of behavior steps necessary for convergence of learning, 1200, is not large taking account of this frequency. The average number of behaviors converged to about 2.2. This means very good performance because the average number by the optimal policy is just a little less than 2.[*9] After convergence (9001–10000 behavior steps), the rover could reach the goal within two behavior steps in 87% of trials.

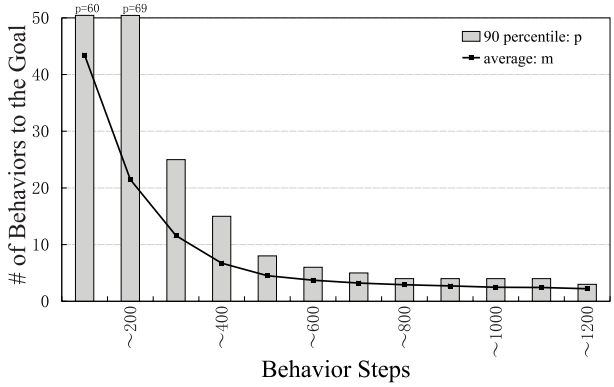


Fig. 6 The average number of behaviors to the goal

By this result, it was verified that STNS can acquire a good situation representation and a good behavior policy efficiently in this simple problem.

7.2 Experiment 2: Flexibility Test

To test the flexibility of STNS, we conducted two experiments on computer simulation. One is a flexibility test to sensor trouble and the other is a flexibility test to actuator trouble. In both experiments, we prepared an STNS after 10000 behavior steps in the above experiment (shown in Fig. 5a), changed the environment, and then let the STNS continue learning in the new environment. As changes of environment, we rotated the direction of the goal sensor

[*9] If the front or the rear of the rover looks just toward the goal, the number of necessary behaviors is 1. In the other directions, it is 2.

in the former test, and made the revolution rate of the left wheel lower in the latter test.[*10].

[1] Flexibility Test to Sensor Trouble

We conducted 10 trial sets in each case of 10, 15, 20, 25, 90, and 180 degrees rotation of the goal sensor. Fig. 7 shows a typical result in case of 15 degrees rotation. As shown in this figure, a good situation representation and a good behavior policy were acquired adapting to the trouble.

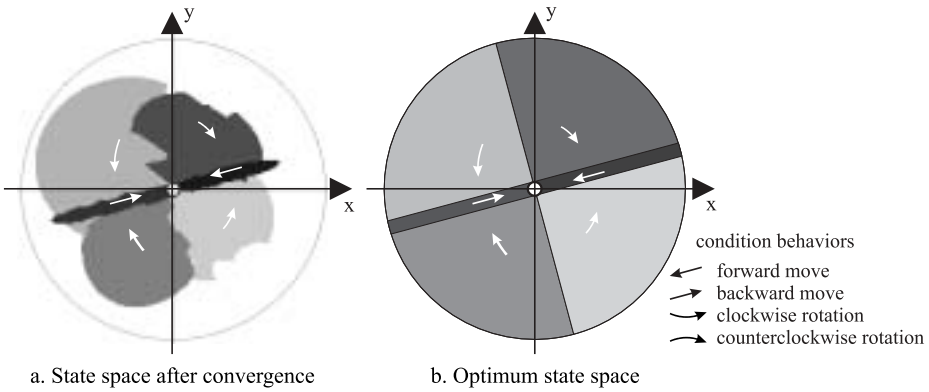


Fig. 7 A result of 15 degrees rotation of the goal sensor

Fig. 8 shows the average number of behaviors to the goal after the trouble in the cases of 10, 15, 20, and 25 degrees rotation comparing with the learning from the beginning (Experiment 1). All trial sets were partitioned into blocks of 250 behavior steps, and the number of behaviors was averaged in each block. As shown in this figure, STNS can adapt flexibly to the trouble in case of rotation within 15 degrees. In this case, all situations transformed their shapes adaptively without being eliminated. In case of rotation over 20 degrees, STNS cannot adapt flexibly. It can, however, be said that the learning is stable since the learning converged to the same performance after a delay of at most 1000 behavior steps as compared with learning from the beginning. Also in the cases of 90 and 180 degrees rotation, the learning converged to the same performance after a delay of at most 1000 behavior steps.

[*10] With this trouble, the rover curves to the left in forward movement and backward movement

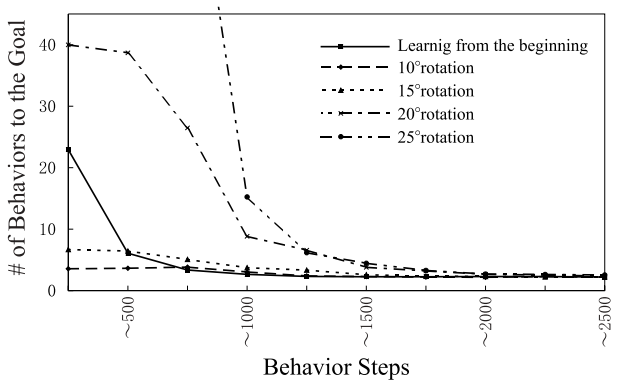


Fig. 8 The average number of behaviors to the goal

[2] Flexibility Test to Actuator Trouble

We conducted 10 trial sets in each case of 5%, 10%, and 15% lower speed of the left wheel. Fig. 9 shows a typical result in case of 15% lower speed. As shown in this figure, a passably good situation representation and a good behavior policy were acquired adapting to the trouble. Situation 8 and 9 in the optimum state space were not extracted in most cases (only two situations were extracted in 10 trial sets). The main reason is that they are located near the edge of the state space, therefore, the density of data is low in their area.

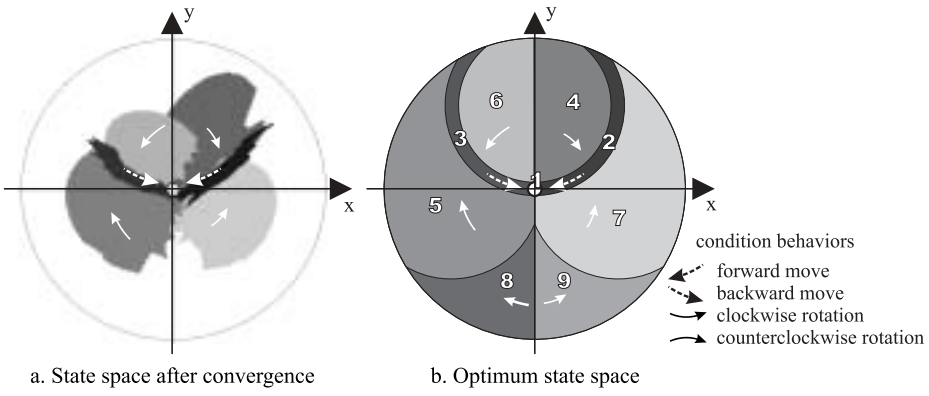


Fig. 9 A result of 15% lower speed of the left wheel

Fig. 10 shows the average number of behaviors to the goal after the trouble. Each graph illustrates the comparison between relearning with the trouble after learning has converged without the trouble and learning with the

trouble from the beginning. All trial sets were partitioned into blocks of 250 behavior steps, and the number of behaviors was averaged in each block. As shown in this figure, STNS can adapt flexibly to the trouble in case of 5% lower speed. In this case, all situations transformed their shapes adaptively without being eliminated. In case of over 10% lower speed, STNS cannot adapt flexibly. It can, however, be said that the learning is stable since the learning converged to the same performance after a delay of at most 1000 behavior steps as compared with learning from the beginning.

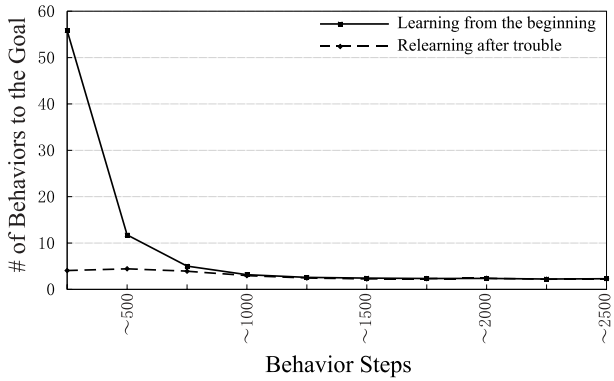
[3] Conclusion of the Two Flexibility Tests

In both experiments, STNS can adapt flexibly to small changes while keeping the high performance and can adapt stably even to big changes. This adaptability is caused by the flexible situation representation. STNS can adapt quickly to small changes by transforming the shapes of situations. In this process, it can keep the high performance by utilizing past experiences. Furthermore, it can adapt not slowly to big changes by eliminating deformed and obstructive situations and starting again from the blank state space. These two types of adaptation can be switched to each other autonomously.

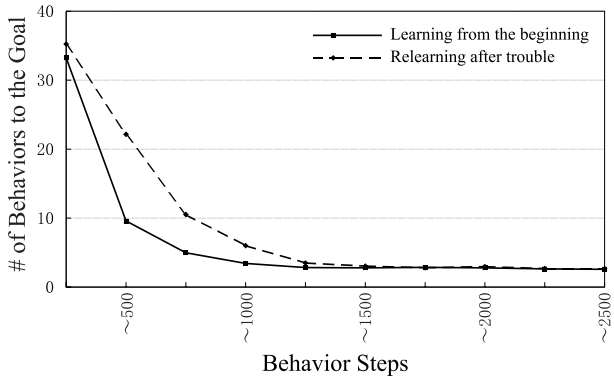
7.3 Experiment 3: Navigation on 8-D Input

The last experiment is a navigation problem on 8-dimensional input. The setting is almost the same with the previous navigation problem on 2-dimensional input. But one obstacle was settled in the room and the goal was fixed to the position shown in Fig. 11a. And we mounted a 6-dimensional obstacle sensor on the rover in addition to the 2-dimensional goal sensor. The obstacle sensor outputs the distance to the nearest obstacle (including the wall) in each of six areas illustrated in Fig. 11b. This problem is difficult because the state space is high dimensional. Furthermore, it is impossible to perform path planning in the state space because the input vector is varied discontinuously often, for example, when the rover passes by a corner of the obstacle.

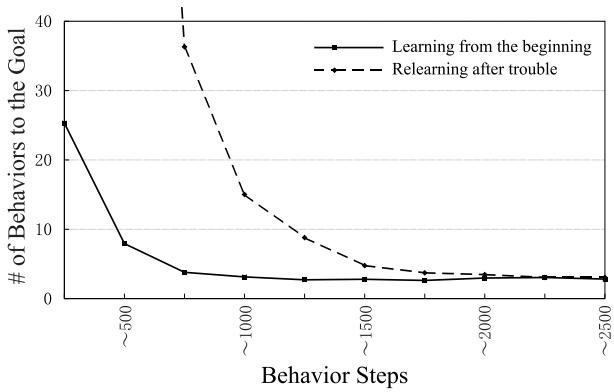
We designed STNS for learning a behavior policy of this problem, and conducted an experiment on computer simulation. We set parameters as follows: $N^R_{init} = N^R_{min} = 9, N^T_{init} = 20, N^T_{min} = 12, r^R_{min} = 5.0, \gamma = 0.7, p_{min} = 0.2$, $n_{max} = 7$. The number of data in the history database is 10000. Both the boundaries of all hyper-ellipsoids and the order of situations are adjusted every 100 behavior steps. Note that the number of positive instances for R-Situation



a. 5% lower speed

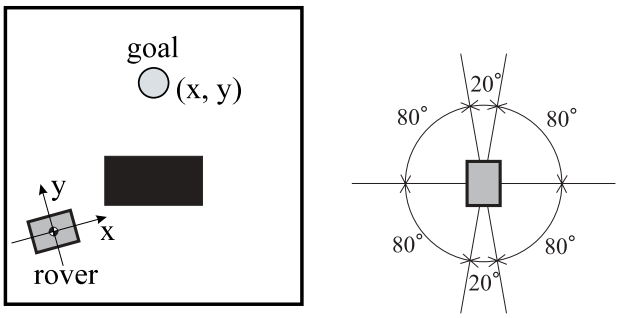


b. 10% lower speed



c. 15% lower speed

Fig. 10 The average number of behaviors to the goal



a. Work space b. 6-D obstacle sensor

Fig. 11 The navigation problem on 8-D input

extraction, $N^R_{init} = 9$, is the minimum for constructing a hyper-ellipsoid in the 8-dimensional state space. This early extraction accelerates learning (see subsection 2.2).

Fig. 12 shows a typical segmentation after enough learning (28000 behavior steps) and the optimum segmentation. The 8-dimensional state space was mapped onto the 2-dimensional work space by fixing the attitude of the rover. As shown in this figure, situations near the goal (situation 2–9) have correspondents in the optimum representation. However, the shapes are distorted as compared with the result of the above 2-dimensional problem. The distortion of the situation representation is proved by the lower success probabilities of behaviors than the 2-dimensional problem. In data of 27001–28000 behavior steps, the average success probabilities of behaviors from R-Situations, T-Situations two behavior steps away from the goal and T-Situations three behavior steps away from the goal were 69.8%, 70.8% and 62.0%, respectively. In the areas more than four behavior steps away from the goal, no situation was extracted. The average number of behaviors to the goal was 6.85. This value is not good because the rover can get to the goal within four behaviors from almost all positions in the room.

This low performance is caused by the lowness of the success probabilities of behaviors in the master plan. Because the shapes of situations are distorted, the rover often fails to reach the parent situation according to the master plan and must make another plan. This increases the number of behaviors to the goal. The lowness of the success probabilities of behaviors leads to the fewness

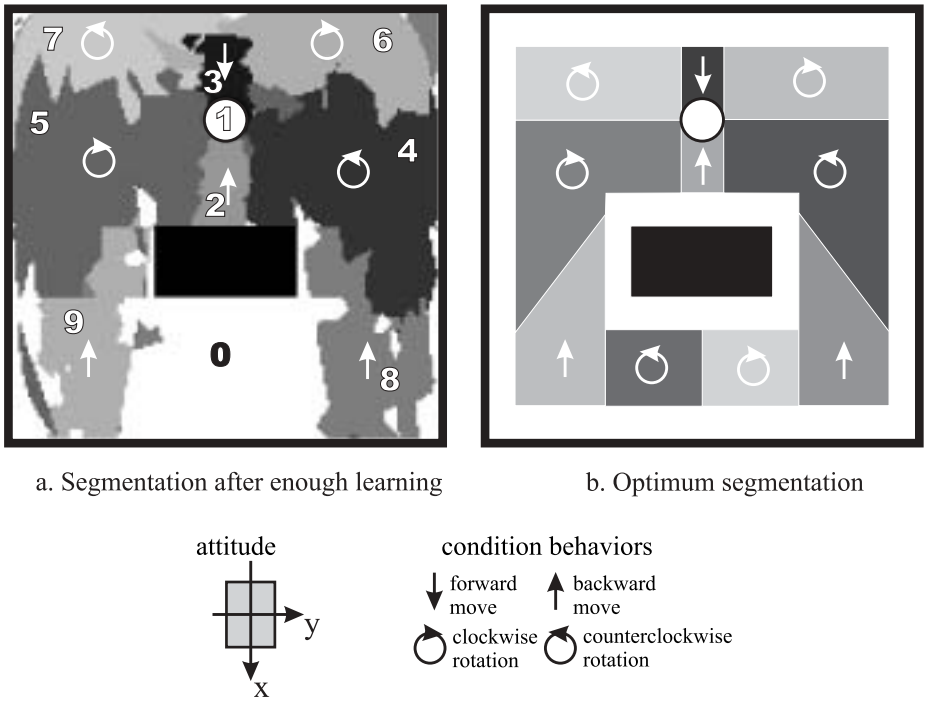


Fig. 12 A segmentation after enough learning and the optimum segmentation

of the positive instances and the lowness of the utilities of situations. When the utility is low, behavior selection becomes diversified because the difference of the expected reward between behaviors becomes small. This diversity leads to the fewness of the positive and negative instances. This fewness leads to the distortion of situations, and a vicious circle is formed as a result. It is difficult to maintain situations in the area far from rewards because both the density of the data and the utilities of the situations are low. In order to sever this vicious circle, the following four approaches are promising:

- Improving the situation representation in order to construct a situation with a smooth shape in the high dimensional state space.
- Improving behavior selection and its stopping condition in order to collect high quality data which can represent situations efficiently.
- Preprocessing the perceptual input and making new attributes for axes of the state space in order to reduce dimensions or approximate the shapes of situations to a hyper-ellipsoid.
- Dividing recognition into multiple layers and performing simple recogni-

tion in each layer.

In this experiment, learning converges very slowly. One of the reasons for this is random behaviors without heuristics in the early stage of learning. The rover can arrive at the goal only 1.34 times in every 100 behavior steps by random behaviors. Therefore, many behaviors (on average, 1034.8 behaviors) are needed for extracting the first situation in the blank state space. Some kind of heuristics, for example greedy hill-climbing method, may accelerate learning.

7.4 Comparison with off-line learning

In the above three experiments, STNS can adapt to the environment autonomously without switching phases. In Experiment 2, it can adapt flexibly to small changes while keeping the high performance. This autonomous adaptability cannot be realized by off-line learning.

The on-line learning in STNS is superior to off-line learning from random behaviors also in efficiency. We will compare these two types of learning using the result of Experiment 1. In this experiment, learning has just converged after 1200 behavior steps. At that time, the average number of positive instances of all R-Situations and all T-Situations in the state space are 225.5 and 212.7 respectively. By random behaviors, the rover takes $225.5 \times 48.09 = 10844$ behaviors on average to collect positive instances of R-Situations because it takes 48.09 behaviors on average to arrive at the goal once. Furthermore, the rover takes $212.7 \times 4 = 850$ behaviors on average to collect positive instances of T-Situations because it can get a positive instance once in about 4 behaviors in a T-Situation. In STNS, the rover took 967.9 behaviors on average in 1200 behavior steps. Thus, off-line learning needs behaviors 12.1 times as many as the on-line learning in STNS in order to collect positive instances as many. It follows from this comparison that STNS can perform categorical learning more efficiently than off-line learning from random behaviors.

7.5 Comparison with the grid representation

In the domain of reinforcement learning, the most usual discrete representation is the grid representation. Fig. 13 shows examples of this representation of the goal sensor space. In Fig. 13a, the space is discretized on the polar coordinate system. In Fig. 13b, the space is discretized on the orthogonal coordinate system. Both representations are discretized by us at equal intervals by which the optimum state space shown in Fig. 5b can be represented roughly.

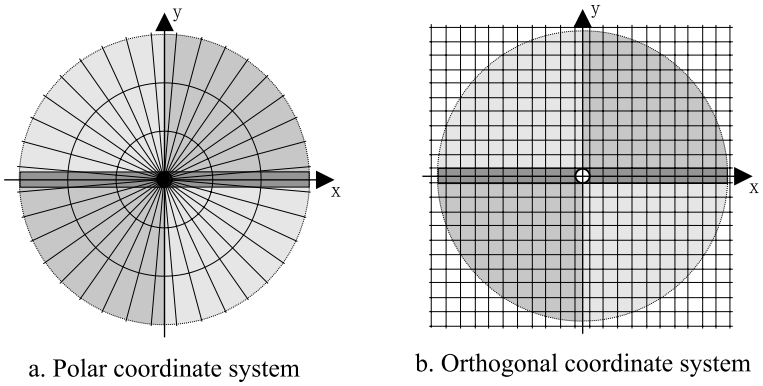


Fig. 13 The grid representation of the goal sensor space

These representations have the following four weak points:

1. They have many states. In the navigation problem on 8-dimensional input, if the obstacle sensor is discretized into three levels (near, middle, and far), the number of states is 78732 with the representation shown in Fig. 13a, and 321489 with the representation shown in Fig. 13b. These numbers are very high. This causes explosion of learning time.
2. The system shifts to various states from the same state by the same behavior. On the representation shown in Fig. 13a, straight movings cause dispersion of reached states. On the representation shown in Fig. 13b, rotations cause dispersion of reached states. This dispersion delays convergence of learning very much.
3. They cannot represent the accurate shapes. The representations are too large-meshed to represent various situations such as shown in Fig. 7 and Fig. 9.
4. They are fragile to the change of environment because the utilities of all states need to be relearned.

The situation representation in STNS is superior to them in points of both abstractness and fineness just enough for task execution.

§8 Conclusion

We have proposed STNS, an architecture for performing both categorical learning and behavioral learning in parallel with task execution in order to solve the problem how to adapt autonomously to various environments and

their changes. In this architecture, categorical learning is improved in autonomy, adaptability, and efficiency by cooperating with behavioral learning. In order to realize this cooperation, we developed a new flexible state representation, the bitten hyper-ellipsoid representation, and five autonomous operations for maintaining the state representation. It was shown in computer simulations that an agent with this architecture can adapt to the task efficiently and continue task execution without breaking down against changes in the environment. We would now like to go on develop this type of parallel learning in both state representation and behavior rules for the purpose of applying it to real problems which have a higher dimensional state space and need more behaviors to achieve the task.

References

1) Asada, M., Noda, S. and Hosoda, K., "Action-based sensor space categorization for robot learning," in *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 96)*, pp. 1502–1509, IEEE, 1996.

2) Barto, A. G., Sutton, R. S. and Anderson, C. W., "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, *13*,*5*, pp. 834–846, 1983.

3) Cohen, P. R. and Feigenbaum, E. A., editors, *The Handbook of Artificial Intelligence*, volume 3, chapter 14, pp. 323–511, William Kaufmann, 1982.

4) Doya, K., "Temporal difference learning in continuous time and space," *Advances in Neural Information Processing System* (Touretzky, D. S., Mozer, M. C. and Hasselmo, M. E. ed.), volume 8, MIT Press, 1996.

5) Harnad, S., "Category induction and representation," in *Categorical Perception: The Groundwork of Cognition* (Harnad, S. ed.), chapter 18, Cambridge University Press, 1987.

6) Ishiguro, H., Sato, R. and Ishida, T., "Robot oriented state space construction," in *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 96)*, pp. 1496–1501, IEEE, 1996.

7) Kröse, B. J. A. and Eecen, M., "A self-organizing representation of sensor space for mobile robot navigation," in *Proceedings of the 1994 IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS 94)*, pp. 9–14, IEEE, 1994.

8) Kröse, B. J. A. and Dam, J. W. M. van, "Adaptive state space quantisation for reinforcement learning of collision-free navigation," in *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 92)*, pp. 1327–1332, IEEE, 1992.

9) Lin, L., "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, *8*, *3/4*, pp. 293–321, 1992.

10) Martín, P. and Millán, J. del R., "Reinforcement learning of sensor-based reaching strategies for a two-link manipulator," in *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 96)*, pp. 1345–1352, IEEE, 1996.

11) Mohri, T., "Nearest Neighbor Rule and Memory-Based Reasoning," *Journal of Japanese Society for Artificial Intelligence*, *12*, *2*, pp. 188–195, 1997. (in Japanese)

12) Moore, A. W., "Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces," in *Proceedings of the Eighth International Workshop on Machine Learning (ML91)*, pp. 333–337, 1991.

13) Moore, A. W. and Atkeson, C. G., "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces," *Machine Learning*, *21*, *3*, pp. 199–233, 1995.

14) Nakamura, T., Takamura, S. and Asada, M., "Behavior-based map representation for a sonar-based mobile robot by statistical methods," in *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 96)*, pp. 276–283, IEEE, 1996.

15) Russell, S. and Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice Hall, 1995.

16) Simons, J., Brussel, H. Van, Schutter, J. De and Verhaert, J., "A self-learning automaton with variable resolution for high precision assembly by industrial robots," *IEEE Transactions on Automatic Control*, *27*, *5*, pp. 1190–1113, 1982.

17) Sutton, R. S., "Learning to predict by the method of temporal differences," *Machine Learning*, *3*, *1*, pp. 9–44, 1988.

18) Sutton, R. S., "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Proceedings of Neural Information Processing Systems (NIPS 95)*, 1995.

19) Takahashi, Y., Asada, M., and Hosoda, K., "Reasonable performance in less learning time by real robot based on incremental state space segmentation," in *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 96)*, pp. 1518–1524, IEEE, 1996.

20) Tesauro, G., "Practical issues in temporal difference learning," *Machine Learning*, *8*, *3/4*, pp. 257–277, 1992.

21) Unemi, T., "Instance-based reinforcement learning method," *Journal of Japanese Society for Artificial Intelligence*, *7*, *4*, pp. 697–707, 1992. (in Japanese)

22) Yairi, T., Nakasuka, S. and Hori, K., "Autonomous state abstraction from heterogeneous and redundant sensor information," *Journal of Japanese Society for Artificial Intelligence*, *14*, *4*, pp. 667–678, 1999. (in Japanese)

23) Yamada, S., "Reactive planning with uncertainty of a plan," in *Proceedings of The Third Annual Conference on AI, Simulation and Planning in High Autonomy Systems*, pp. 201–208, 1992.