Available online at www.sciencedirect.com
SCIENCE DIRECT®
Engineering Applications of Artificial Intelligence 17 (2004) 577–588
Engineering Applications of
ARTIFICIAL
INTELLIGENCE
www.elsevier.com/locate/engappai

Adaptive state space partitioning for reinforcement learning

Ivan S.K. Lee, Henry Y.K. Lau*

Department of Industrial and Manufacturing Systems Engineering, The University of Hong Kong, Pokfulam, Hong Kong

Received 7 February 2003; accepted 6 August 2004

Abstract

The convergence property of reinforcement learning has been extensively investigated in the field of machine learning, however, its applications to real-world problems are still constrained due to its computational complexity. A novel algorithm to improve the applicability and efficacy of reinforcement learning algorithms via adaptive state space partitioning is presented. The proposed temporal difference learning with adaptive vector quantization (TD-AVQ) is an online algorithm and does not assume any a priori knowledge with respect to the learning task and environment. It utilizes the information generated from the reinforcement learning algorithms. Therefore, no additional computations on the decisions of how to partition a particular state space are required. A series of simulations are provided to demonstrate the practical values and performance of the proposed algorithms in solving robot motion planning problems.

Keywords: Reinforcement learning; Nearest neighbor quantizer; State space partitioning; Peg-in-hole; Navigation

1. Introduction

Reinforcement learning (RL) deals with the problem encountered by an autonomous agent learning to achieve a goal through interactions with its environment. This learning method uses the experience accumulated by the agent to improve the performance index with respect to a particular task. Applications of RL methods abound, mostly in the fields of game playing (Tesauro, 1992, 1994), robotics (Riedmiller, 1996), scheduling (Zhang and Dietterich, 1996) and inventory control (Mahadevan et al., 1997). Although the convergence property of RL has been widely investigated by machine learning researchers (Sutton, 1988; Watkins and Dayan, 1992; Dayan, 1992; Dayan and Sejnowski, 1994; Jaakkola et al., 1994; Tsitsiklis, 1994; Bertsekas and Tsitsiklis, 1996), its applications to practical problems are still constrained by the curse of dimensionality (Bellman, 1957). This constraint is due to the exponential increase in the number of admissible states with the dimensionality of state space. As a result, practical applications, especially in solving real-time control problems with RL methods, are hindered by the required computational workload.

Since most theoretical results of RL algorithms consider problems with discrete and finite state spaces, this investigation studies the discretization process that transforms an RL problem with continuous state space into one with discrete state space. A novel algorithm employing adaptive vector quantization (AVQ) is developed to partition the state space incrementally as the autonomous agent interacts with its environment. It does not assume any a priori knowledge with respect to the learning task and environment. The AVQ algorithm utilizes the information generated from the RL algorithms and, therefore, does not require additional computations on the decisions of how to partition a particular state space.

A review of reinforcement learning is presented in Section 2. The proposed AVQ algorithm for the

*Corresponding author. Tel.: +852-2857-8255; fax: +852-2858-6535.
E-mail address: hyklau@hkucc.hku.hk (H.Y.K. Lau).

0952-1976/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2004.08.005

adaptive partitioning of the state space is described in detail in Section 3. The characteristics and performance of the AVQ algorithm are investigated via a series of navigation tasks in Section 4. Finally, the application of AVQ to the peg-in-hole task is presented in Section 5 and a conclusion is drawn in Section 6.

2. Reinforcement learning

A discrete time-, action- and state-space reinforcement-learning model is presented in this section. In a typical reinforcement learning model (Kaelbling et al., 1996), a learning agent is connected to its environment via perception and action (Fig. 1). The state of the environment at time t is denoted by $\mathbf{x}_t \in S$, where S is the set of admissible states. The agent selects an action $\mathbf{u}_t \in U(\mathbf{x}_t)$, where $U(\mathbf{x}_t)$ is the set of admissible actions in state $\mathbf{x}_t$, according to a control policy. The control policy $\mu_t : S \to U$ specifies each admissible action as a function of the observed state. At the next time step $(t+1)$, after executing the action, it receives an immediate reward according to the reward function $g(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$.

The aim of the learning agent is to develop an optimal control policy, and its associated optimal state trajectory, with respect to a particular measure. The system dynamics and constraints are not known a priori. Instead, the actual system is directly accessible to the learning agent. Therefore, the agent must actively apply perturbation to the system in order to acquire the necessary information.

Solving a reinforcement learning problem involves computational algorithms of searching for the optimal policy μ^* that maximizes the total expected reward. This is achieved by estimating the action value function $Q^\mu(\mathbf{x}, \mathbf{u})$, $\forall \mathbf{x} \in S$ and $\forall \mathbf{u} \in U(\mathbf{x})$, which is the total expected reward of starting at state $\mathbf{x}$, taking action $\mathbf{u}$, and thereafter following policy μ. For an infinite horizon problem, where the terminal time T approaches infinity, the action value function is given by

$$Q^\mu(\mathbf{x}, \mathbf{u}) = \lim_{T\to\infty} E\left\{\sum_{t=0}^{T-1} \gamma^t g(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})\right\}, \tag{1}$$

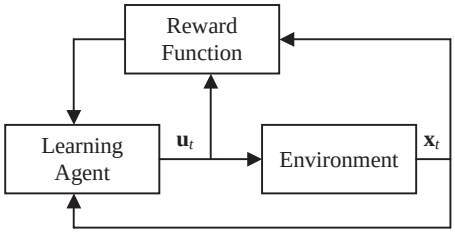

Fig. 1. The reinforcement learning model.

where $0 < \gamma \leqslant 1$ is the discount factor. The optimal action value function $Q^*(\mathbf{x}, \mathbf{u})$ is defined as

$$Q^*(\mathbf{x}, \mathbf{u}) = \max_\mu Q^\mu(\mathbf{x}, \mathbf{u}), \quad \forall \mathbf{x} \in S,\ \mathbf{u} \in U(\mathbf{x}). \tag{2}$$

In fact, the estimation of action value function plays a critical role in all reinforcement learning problems. It is because once the optimal action value function is found, the corresponding optimal policy can be determined with relatively little computation by adopting a greedy policy with respect to the optimal action value function. A greedy control policy is defined as one that always chooses actions that lead to the maximum next-state value. The value of any state is the largest of all action values for that particular state, i.e.,

$$V^\mu(\mathbf{x}) = \max_{\mathbf{u}\in U(\mathbf{x})} Q^\mu(\mathbf{x}, \mathbf{u}), \quad \forall \mathbf{x} \in S. \tag{3}$$

All practical RL algorithms aim at estimating the state value, i.e., the total expected reward of being at a particular state and following a particular control policy. Since an analytical model of the system or environment is not available, any algorithms that try to predict the state value must rely on the training data generated from the actual system. A class of incremental learning procedures for this particular prediction purpose is called temporal difference (TD) method (Sutton, 1988).

Temporal credit assignment problem is relevant to any planning problems in which a series of actions are taken before attaining the terminal state (Sutton and Barto, 1998). One way to tackle the temporal credit assignment problem is to incorporate *eligibility trace* into TD methods. The eligibility $\zeta(\mathbf{x})$ of a state $\mathbf{x}$ is the degree to which it has been visited in the recent past. With eligibility trace, one can distribute the immediate reinforcement signal to all the states that have been recently visited, according to their eligibility. The eligibility trace can be updated online with the following update rule:

$$\zeta(\mathbf{x}) := \begin{cases} \lambda\zeta(\mathbf{x}) + 1 & \text{if } \mathbf{x} = \text{current state}, \\ \lambda\zeta(\mathbf{x}) & \text{otherwise.} \end{cases} \tag{4}$$

Denoted by TD(λ), temporal difference with eligibility trace converges faster for large λ (Dayan et al., 1994). Although TD(λ) is a powerful algorithm of solving learning tasks autonomously (Tesauro, 1994; Riedmiller, 1996; Zhang and Dietterich, 1996; Mahadevan et al., 1997), it applies to problems with discrete state- and action-spaces. As a result, any problems in continuous domain must be transformed into discrete domain before TD(λ) can be applied. The next section discusses such transformation process and presents the adaptive vector quantization (AVQ) algorithm for the partitioning of continuous state spaces.

3. Adaptive state space partitioning with vector quantization

Solving a reinforcement learning problem with TD learning methods relies on the estimation of the optimal value function. In general, the optimal value function is not analytical and, hence, cannot be represented by a closed form analytical function. For the purpose of computation, the look-up table method provides an avenue to store a general function. For tabular representation of the value function, one would represent the value function as an array of real numbers indexed by the states of the problem. Since most real-life problems involve continuous state variables, a discretization process is essential to the application of TD learning methods. In essence, the discretization process partitions the continuous state space into a finite number of subsets. Each individual subset represents a discrete state of the agent.

One of the most appealing features of a reinforcement learning agent is autonomy. It must be capable of achieving the goal, defined in terms of the reward function, with minimum supervision from human operators. Ideally, it must also be able to adapt to the changing environment in order to fulfill the designated task. Therefore, the partitioning of the state space should be performed as part of the learning task, instead of being fixed at the start of the learning process. In this way, the agent learns a suitable partitioning for a particular task as it gains more information from the environment.

The objective of adaptive state space partitioning is to develop an efficient online algorithm for the discretization of a continuous state space. Concurrently, the TD learning algorithms are applied to this discretized state space. Hence, both the discretization process and the TD learning algorithm are carried out simultaneously based on the real-time sensory feedback signals gathered as the agent interacts with its environment.

The design of a discretization mapping plays a critical role in solving practical reinforcement learning problems with TD methods. The convergence property of TD methods with state space discretization is of prime importance. Moreover, computational cost is also an important issue if TD methods are to be employed for real-time robot control. The control loop frequency of a robot controller influences its overall performance to a large extent. It is often the case that high-level planning or intelligent control routines are implemented in parallel with the low-level, but equally important, PID control laws. In other words, the controller is required to handle additional computational operations due to the intelligent control sub-routines. Therefore, any online planning sub-routines should not induce excessive computations otherwise the servo rate would be comprised. In the light of this concern, the computational operations required for the implementation of discretization mapping must be minimized. The nearest neighbor vector quantizer is a favorable candidate for this purpose.

3.1. Nearest neighbor vector quantization

Vector quantization is a generalization of scalar quantization to that of a vector. It is a widely employed encoding technique in signal compression for storage and telecommunication (Gray, 1984). Nearest neighbor vector quantization is a method of partitioning the state space into a set of disjoint regions.

A nearest neighbor quantizer, also called Voronoi quantizer, ψ, maps a vector, $\mathbf{x} \in \Re^n$, where $\Re$ is the set of real number, onto a finite set of codewords (also called codebook vectors), $C = \{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \ldots, \mathbf{c}_m\}$ and $\mathbf{c}_i \in \Re^n$ for $i = 1, 2, \ldots, m$, i.e., $\psi : \Re^n \to C$ where $C \subset \Re^n$. In essence, this mapping divides the n-dimensional real space $\Re^n$ into m disjoint regions called Voronoi cells R_i for $i = 1, 2, \ldots m$, i.e.,

$$R_i = \{\mathbf{x} \in \Re^n : \psi(\mathbf{x}) = \mathbf{c}_i\}. \tag{5}$$

The union of all the Voronoi cells corresponds to the domain of the mapping, , $\cup_i R_i = \Re^n$and $R_i \cap R_j = 0$, for $i \neq j$. If the member vectors of the codebook are determined, the mapping, $\psi(\mathbf{x})$, is defined by the nearest neighbor rule,

$$R_i = \{\mathbf{x} : ||\mathbf{x} - \mathbf{c}_i|| \leqslant ||\mathbf{x} - \mathbf{c}_j||\}, \quad \forall i \neq j. \tag{6}$$

The major advantage of nearest neighbor vector quantizer is that the partition is completely determined by the set of codewords C and the mapping ψ may be implemented with a few computational operations. By mapping any particular continuous state vector, $\mathbf{x}$, into one of the codewords, $\mathbf{c}_i$, the total number of admissible states is greatly reduced.

Designing a nearest neighbor vector quantizer amounts to defining the set of codewords. If the optimal value function is known a priori, each region defined by the corresponding codeword should include all the states with approximately the same value. However, rather than providing a closed form solution, TD learning is a computational algorithm that iteratively improves its estimate of the value function. Therefore, the codewords have to be defined and adjusted along with the TD operations to reflect the most recent estimate of the value function. Two fundamental codeword manipulations are considered in the next section, namely, the appending and merging of codewords.

3.2. Adaptive vector quantization

3.2.1. Defining codewords

To partition the state space is equivalent to approximating the value function with piecewise constant

functions because any perception vectors that fall into a particular cell is viewed as identical. In other words, they share the same action value profile. The aim of the proposed partitioning algorithm is to group together states with similar action value profile and next state transition probabilities. This notion is implemented in the following operations.

First, a new quantity called the accumulated reward (*accReward*) must be defined. The accumulated reward with respect to a particular action is the sum of the total rewards received by continuously taking the same action within a particular cell.

At the beginning, the codebook is initialized to consist of just one codeword, which may, for example, correspond to the initial or goal state. In other words, the learning agent views the entire state space as a homogeneous region when no a priori knowledge is provided. Subsequently, the appending of new codewords to the Voronoi quantizer is performed based on two criteria:

1. The accumulated reward signal (*accReward*) exceeds a certain threshold, χ.
2. The Euclidean distance between the newly appended and its nearest neighbor codeword must be greater than a threshold value which corresponds to the minimum resolution, Δ.

The first criterion limits the variation of action value within any particular cell. The second criterion maintains a predefined minimum resolution for practical implementation reasons.

3.2.2. Merging codewords

As the agent continues to explore its environment, the number of codewords would increase. It is, therefore, essential to merge any similar codewords in order to keep the size of the codebook to a minimum without severely degrading its capability of searching for the optimal control policy. Since the aim of state space partitioning is to facilitate the search for an optimal control policy, it is logical to suggest that the optimal control action of every constituent generic state contributing to a particular aggregate region or Voronoi cell (R_i) should be similar. However, it is very unlikely that the optimal or even the near optimal action value function and the associated optimal policy be available with a few iterations of TD algorithms. Theoretically, it converges to the optimal solution in an infinite number of iterations. As a result, state or vector aggregation based upon the optimal action alone is impractical.

TD algorithms maintain an estimate of the action value profile, $Q(\mathbf{x},\mathbf{u})\ \forall \mathbf{u} \in U(\mathbf{x})$, for every encountered state. Essentially, this profile shows the merit of every admissible action relative to each other of the same state. This action value profile provides a better similarity measure among various states. Hence, states with similar action value profiles should be combined to form an aggregate state. The merging criterion of a pair of nearest neighbor codewords is.

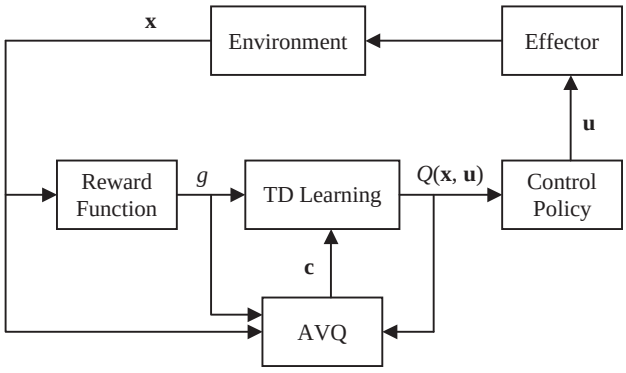

Fig. 2. The TD-AVQ control scheme.

Any pair of nearest neighbor codewords are merged into one if the mean square difference between the action value profiles is less than a threshold, ρ.

Adaptive vector quantization is designed to implement the online partitioning of state space according to the action value profiles, which are estimated by RL algorithms. In other words, the partition is dependent upon the RL algorithms employed. The temporal difference learning with adaptive vector quantization (TD-AVQ) control scheme is shown in Fig. 2 and the complete algorithm is shown in Fig. 3. In the following sections, a number of simulated experiments are performed to demonstrate the properties and efficacy of the TD-AVQ algorithm.

4. Navigation tasks

This section elucidates the TD-AVQ algorithm via a series of maze navigation tasks (Lau et al., 2002). The maze navigation task is simulated with the Nomad 200 (Nomadic Technologies, Inc., 1997) mobile robot simulator (Fig. 4). The Sensus 300 proximity system, which has 16 sonar and infrared ranging sensors providing 360° coverage, enables the mobile robot to detect any obstacles around it. The perception state of the mobile robot is defined to be the x- and y-coordinates of its current position, i.e., $\mathbf{x}=[x\text{-coordinate}, y\text{-coordinate}]$. This perception state is mapped to a member of the codebook, C, for subsequent TD(λ) learning. The admissible action of the mobile robot is one unit movement in one of the four compass directions, i.e., $U = \{\text{North, East, South, West}\}$. The control policy is derived based on the current estimate of the action value function and an undirected exploration strategy called ε-greedy policy (Thrun, 1992). In brief, the ε-greedy policy selects action stochastically and the probability distribution of the selected action is

Define accumulated reward threshold (χ) and minimum resolution (Δ)
Initialize the first codeword $\mathbf{c}_1$ (e.g. $\mathbf{c}_1$ = initial state)
Initialize $Q(\mathbf{c}_1, \mathbf{u}) = 0, \forall \mathbf{u}$
Initialize eligibility trace $\zeta(\mathbf{c}_1, \mathbf{u}) = 0, \forall \mathbf{u}$
accReward := 0
Initialize $\mathbf{x}_t$, $\mathbf{u}_t$

Repeat
execute action $\mathbf{u}_t$
observe immediate reward g and next state $\mathbf{x}_{t+1}$
determine activated codeword $\mathbf{c}_{t+1}$ $\psi(\mathbf{x}_{t+1})$ using the nearest neighbor rule:
$\| \mathbf{x}_{t+1} - \mathbf{c}_i \| \leq \| \mathbf{x}_{t+1} - \mathbf{c}_j \|, \forall j \neq i$

determine $\mathbf{u}_{t+1}$ for $\mathbf{c}_{t+1}$ using the ε-greedy policy derived from $Q(\mathbf{c}_{t+1}, \mathbf{u})$

if $\mathbf{c}_{t+1} = \mathbf{c}_t$
accReward += g
if (*accReward* > χ and $\|\mathbf{c}_t - \mathbf{x}_{t+1}\| > \Delta$)
append the new codeword $\mathbf{c}_{new} := \mathbf{x}_{t+1}$
initialize and append $Q(\mathbf{c}_{new}, \mathbf{u})$ and $\zeta(\mathbf{c}_{new}, \mathbf{u}), \forall \mathbf{u}$
accReward := 0
$\mathbf{c}_{t+1} := \mathbf{c}_{new}$
else
$\mathbf{u}_{t+1} := \mathbf{u}_t$
end
else
calculate the TD error:
$\tau := g(\mathbf{c}_t, \mathbf{u}_t, \mathbf{c}_{t+1}) + \gamma Q(\mathbf{c}_{t+1}, \mathbf{u}_{t+1}) - Q(\mathbf{c}_t, \mathbf{u}_t)$
update the eligibility trace of current codeword-action pair:
$\zeta(\mathbf{c}_t, \mathbf{u}_t) := \zeta(\mathbf{c}_t, \mathbf{u}_t) + 1$
refine the estimate of action value function using the TD learning rule:
$Q(\mathbf{c}, \mathbf{u}) := Q(\mathbf{c}, \mathbf{u}) + \alpha\tau\zeta(\mathbf{c}, \mathbf{u}), \forall \mathbf{c}, \mathbf{u}$
decay eligibility trace:
$\zeta(\mathbf{c}, \mathbf{u}) := \gamma\lambda\zeta(\mathbf{c}, \mathbf{u}), \forall \mathbf{c}, \mathbf{u}$
accReward := 0
end

$\mathbf{c}_t := \mathbf{c}_{t+1}$ and $\mathbf{u}_t := \mathbf{u}_{t+1}$

until end of trial

Define the codewords merging threshold (ρ)
For all $\mathbf{c}_i, \mathbf{c}_j \in C$

If ($\| \mathbf{c}_i - \mathbf{c}_j \| < \| \mathbf{c}_i - \mathbf{c}_h \|, \forall h \neq i, j$) and $\{ \sum_{\mathbf{u}} \left[Q(i,\mathbf{u}) - Q(i,\mathbf{u}) \right]^2 \leq \rho \}$
append a new codeword $\mathbf{c}_k = (\mathbf{c}_i + \mathbf{c}_j)/2$
initialize and append $Q(\mathbf{c}_k, \mathbf{u}) := [Q(\mathbf{c}_i, \mathbf{u}) + Q(\mathbf{c}_j, \mathbf{u})]/2, \quad \forall \mathbf{u}$
remove $\mathbf{c}_i$, $\mathbf{c}_j$, $Q(\mathbf{c}_i, \mathbf{u})$, $Q(\mathbf{c}_j, \mathbf{u}), \forall \mathbf{u}$
End

Fig. 3. The complete TD-AVQ algorithm.

determined according to the recent estimate of the action value function. With a probability of ε, the mobile robot selects uniformly among all the admissible actions and the action with the highest estimated value will be selected with a probability of $1 - \varepsilon$.

The state space partition and the highest-valued action of each Voronoi cell after 1000 trials are shown in Fig. 5a. A sampled path from a start to the goal position is also shown. By inspection, this partition and the associated optimal actions can direct the mobile robot to the goal position from almost everywhere within the maze. However, the shortest path to destination is not always guaranteed, such as starting in cell 4. This is mainly due to the drawback of

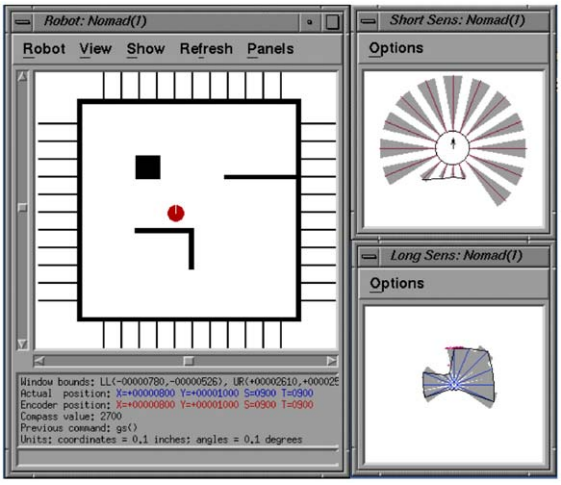

Fig. 4. Nomad 200 Robot simulator.

undirected exploration strategy as discussed in (Thrun, 1992) and the coarse resolution on state space. Nevertheless, this is an inevitable cost of coarsely partitioning the state space. To find the shortest path, one must allow maximum resolution on the position coordinates, which results in more than 100 states. However, the adaptive partitioning algorithm, TD-AVQ, produces only 22 aggregate states. Each Voronoi cell in Fig. 5a represents an aggregate state. In fact, the total number of the codewords depends on the structure of the maze. Given a particular area of the maze, the more barriers there are, the more codewords are required to partition the state space in order to correctly reflect the complexity of the maze. This is reflected by the smaller and greater number of Voronoi cells around the goal position and obstacles where distinctive optimal actions are needed. On the other hand, cells with larger area are observed in the regions of free space where a large number of states share the same optimal action.

As shown in Fig. 5a, there are regions in which following the highest-valued action could not lead to the goal. This is due to the fact that this failed region is not adequately explored. The actual location of each codeword and, hence, the resulting state space partition depends on the current estimate of the action value function to a large extent. With the TD learning algorithm, the value update operation is applied to any state in any order using any information that is obtained by observing the mobile robot performing the task in real time. Therefore, the frequency with which the estimated action value is updated for any state is highly dependent on the exploration strategy.

In order to eliminate any failed regions, every part of the maze must be visited. To support this argument, the navigation task is extended and trained for an extra 200 trials. In order to establish the validity of the partitioning scheme, the starting position is reset within the failed

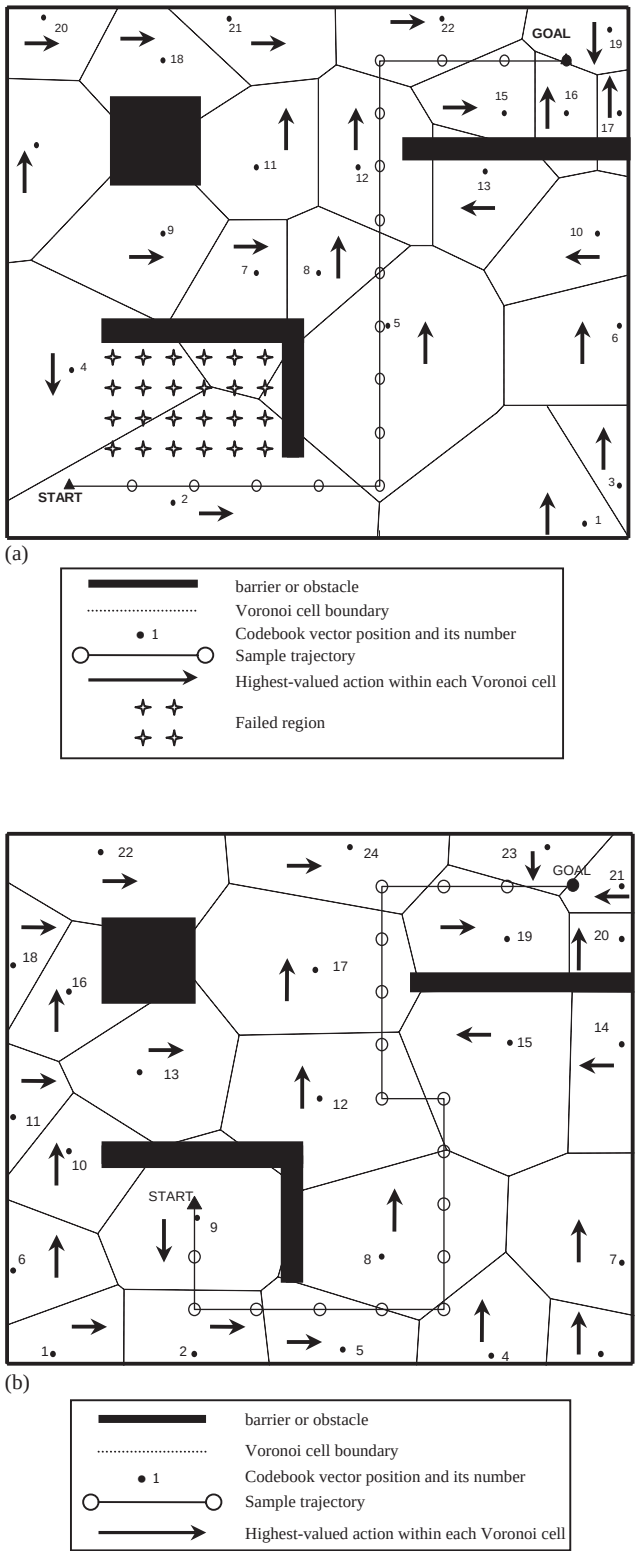

Fig. 5. (a) State space partition on a maze as developed by TD-AVQ after 1000 trials. (b) State space partition on a maze as developed by TD-AVQ after 1200 trials.

region. As shown in Fig. 5b, a new codeword situated inside the failed region is appended to the codebook and the highest-valued action of the region can lead to the

goal. In addition, the positions of other codewords are changed. This reflects the fact that the partitioning scheme is dependent upon the current estimate of the action values. As the estimate of action value function varies when TD learning algorithm is applied, so does the partitioning scheme of the state space.

A second navigation task is performed to show that adaptive state space partitioning facilitates TD learning for model-free path planning problems. The main focus of interest is on problems where there are critical regions in the working environment that require finer resolutions than other non-critical regions. This is demonstrated through a specially designed maze navigation task.

Fig. 6a shows the state space partition of a specially designed maze with critical region as developed by the TD-AVQ algorithm after 5000 trials. By closely examining the partition, the following observations are made. A large number of cells with small area scatter around the hole, along the barrier and the maze boundaries. To the contrary, a few cells with large area cover much of the obstacle-free space.

4.1. Comparison with uniform discretization

To investigate the relative merit of AVQ over uniform discretization, the same navigation task is repeated with uniform state space partition. The same TD methods and parameters are used to solve the navigation task. To adopt a fixed-grid uniform discretization of the state space, the resolution of the entire state space would be determined by the width of the hole on the barrier. This results in nearly 1500 states in total. During the entire learning process, which comprises 10,000 trials with maximum steps of 1000, the mobile robot has never succeeded in reaching the goal line. In other words, it fails completely in this navigation task.

The exploration strategy adopted in this experiment causes the mobile robot to take random and explorative actions earlier in the learning process. The reward function is designed such that there would not be any significant reward except hitting the goal line. Therefore, it is by reaching the goal line that the mobile robot may deduce which actions and states would yield better rewards in the long run. If, however, the mobile robot cannot reach the goal line for once, it would receive the same reward irrespective of what action it takes in any states. To sum up, the mobile robot must reach the goal line at least once before it starts taking exploitative actions. Unfortunately, the probability of passing through the hole and reach for the goal by performing a random walk on the uniformly discretized state space is very small.

In a deterministic environment with N states and m admissible actions for each state, there are m^N possible control policies. In this specially designed maze, the shortest distance from the starting line to the goal line is 60 units. As a result, the agent must take at least 60 correct actions in order to reach the goal. By choosing randomly from the 4 admissible actions in each state,

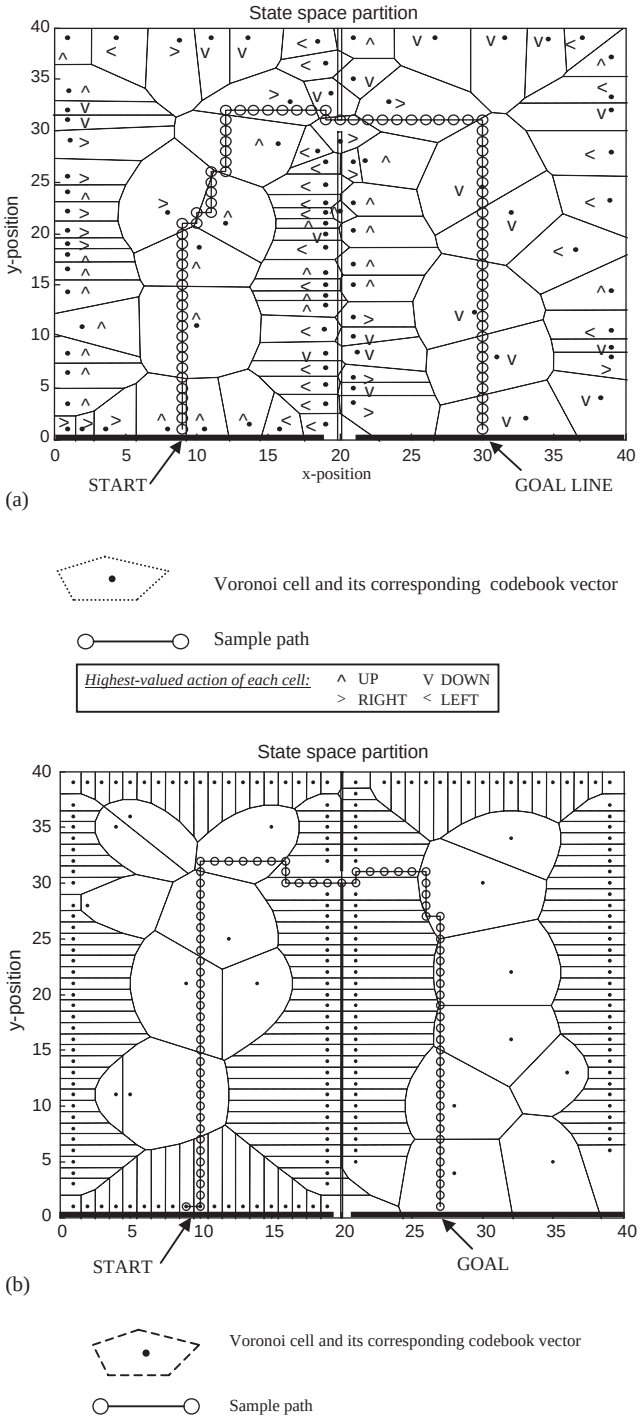

Fig. 6. (a) The state space partition for a maze with critical region as developed by TD-AVQ after 5000 trials. (b) The state space partition for the maze with an opening on the barrier as developed by TD-AVQ *without* merging operations after 5000 trials. The total number of cells is 204 after 5000 trials.

i.e., performing a random walk, the probability of reaching the goal line in 60 steps is $0.25^{60} \approx 0$.

The advantage of adaptive state space partitioning over uniform partitioning is mainly due to the reduction in the number of distinguishable states. While uniform partitioning generates a discrete space with 1500 states, AVQ algorithm produces just over 90 aggregate states (see Fig. 6a), i.e., more than 93% reduction. The benefits are twofold. First, since the entire state space comprises a much smaller number of distinguishable states, the chances of visiting all the states and taking every admissible action in each state are dramatically increased even by adopting random walk exploration strategy. This improves the optimality of the control policy.

Secondly, the number of state transitions, or stages, required to reach the goal is decreased due to the overall reduction in the number of distinguishable states. As shown in Fig. 6a, AVQ partitioning turns this shortest path problem into a 15-stage decision problem (since it visits 15 cells to arrive at the goal), comparing with more than 60 stages for uniform partitioning scheme. This reduction in the number of stages has a profound effect on the operation of TD learning algorithm. The number of possible control policies is reduced exponentially from 4^{60} to 4^{15}. Since the computational complexity of solving a K-stage planning problem with N states and m admissible actions for each state is directly proportional to KmN. By adopting AVQ partitioning algorithm, the computational complexity is significantly decreased due to the reduction in the stage (K) and the number of states (N). For this particular navigation task, the complexity is reduced by 98.5% compared to uniform grid partitioning.

This suggests that a feasible solution for the navigation problem can be found by adopting the AVQ algorithm to facilitate TD learning, whereas fixed-grid uniform partitioning of the state space fails completely. The AVQ algorithm requires the action value profile as the similarity measure for state aggregation. This action value profile is readily available form the TD learning operation. Therefore, one can actually utilize the information generated by the TD learning algorithms to help partitioning of the state space. This, in turn, facilitates the TD learning process itself.

4.2. The effect of merging codewords

To investigate the effect of merging codebook vectors, a control experiment is carried out. The TD-AVQ algorithm (shown in Fig. 3) is used to solve the same navigation problem except that the merging operation is left out. The state space partition so developed is shown in Fig. 6b. By comparing this result with Fig. 6a, the major difference lies in the cells along the boundaries and barriers. With merging operations, cells in those regions are merged. Fig. 7 compares the total number of cells at the end of each trial. While the number of cells quickly rises up to above 200 and levels off without merging, the number goes up and then decreases to below 100 with merging. However, the number of cells contributing to the optimal trajectory is similar for both

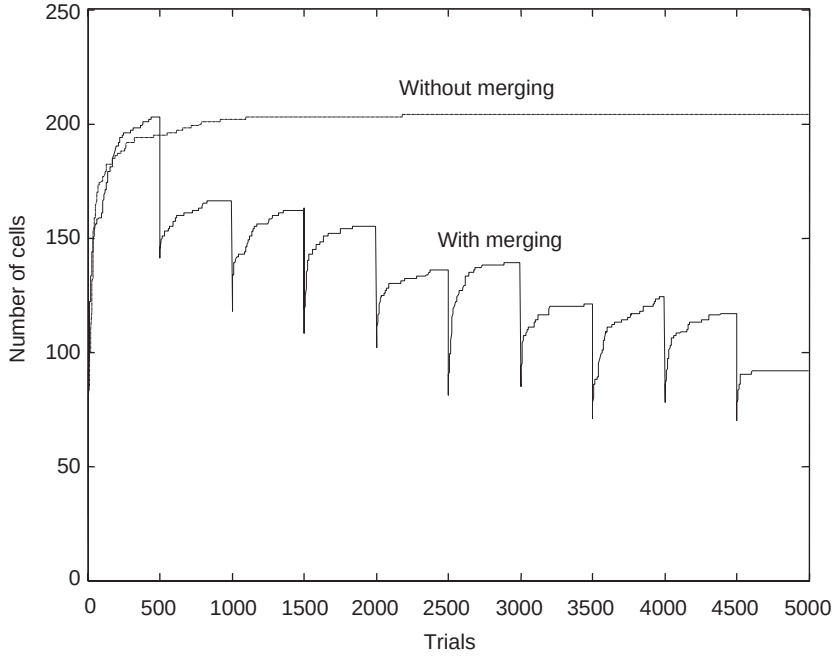

Fig. 7. The effect of codeword merging operation on the number of Voronoi cells. This figure shows that without merging routine, the number of cells rises up quickly to around 200 and then levels off. With codewords merging routine, the total number of cells gradually drops back to below 100 after rising up to 200 at the beginning. Codewords merging operation is performed every 500 trials.

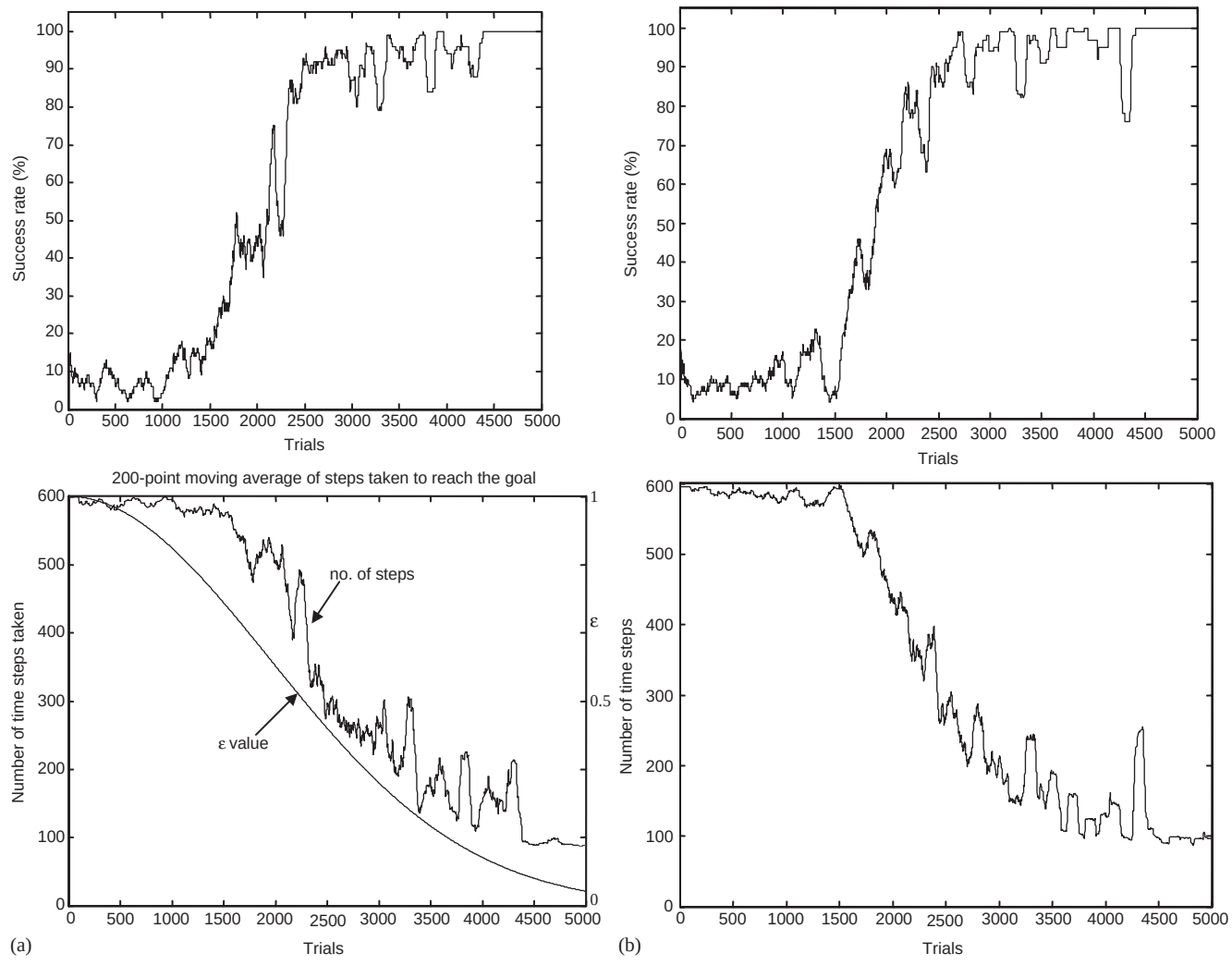

Fig. 8. (a) Learning curves for TD-AVQ in solving the navigation problem. (b) Learning curves for TD-AVQ *without* merging in solving the navigation problem.

cases. Since the merging operation is performed between trials, it does not add to the workload in the real-time task.

Fig. 8 shows the learning curves for TD-AVQ for both with and without codeword merging operations. In terms of learning capability, both variants perform equally well.

5. Peg-in-hole task

To demonstrate the efficacy of adaptive path planning in practical problems, the TD-AVQ algorithm is used to solve a two-dimensional version of the peg-in-hole insertion task in a simulated environment (Lee and Lau, 2000; Lau and Lee, 2001). In particular, the TD-AVQ algorithm is employed to generate a reference trajectory for the robot controller without any a priori information regarding the structure of the entire workspace.

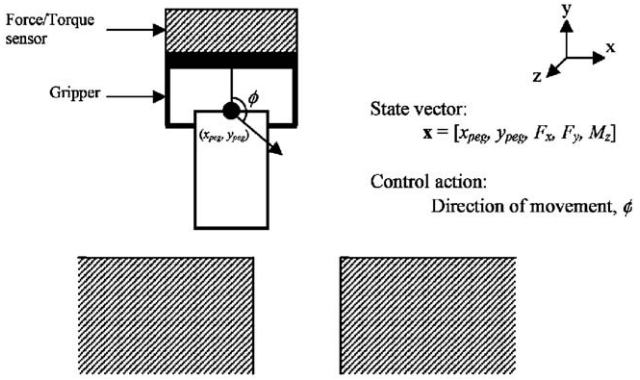

Fig. 9. The peg-in-hole task.

With respect to Fig. 9, a set of feedback signals are provided to the TD-AVQ trajectory planner. It

comprises the position of the manipulator (x_{peg}, y_{peg}) and the force/torque signals (F_x, F_y, M_z) as sensed by a wrist-mounted tactile sensor. These signals constitute a perception state vector $\mathbf{x}=[x_{robot}, y_{robot}, F_x, F_y, M_z]$ characterizing the state of the manipulator at any moment. At the start of each time step, the current perception state vector is fed to the planner/controller. The TD-AVQ trajectory planner computes a reference motion command indicating the direction of movement with respect to the center of the end-effector, ϕ. The set of admissible actions is $U = \{0^\circ, 45^\circ, 90^\circ, 135^\circ, 180^\circ, 225^\circ, 270^\circ, 315^\circ\}$. The manipulator moves with constant speed in the direction given by ϕ. The peg is 8 units long and 4 units wide, while the hole is 4 units wide. Thus, the clearance between the peg and the hole is zero. The dimension of the workspace is 64 units wide and 30 units high. The ε-greedy exploration strategy is adopted in this experiment.

After 10,000 training runs, the position space partition for the peg-in-hole task is shown in Fig. 10. Since the state vector comprises 5 state variables, a state space partition cannot be drawn. However, the position space is a close approximation to the state space since the force/torque variables acquire non-zero values around the hole only.

A learning curve showing the average insertion time achieved by the controller is shown in Fig. 11. In the beginning, the learning agent does not possess any information regarding the geometry of the workspace and the exact position of the hole. Therefore, the behavior of the learning agent is biased towards exploration by choosing the value of ε to be close to 1 at the beginning of the training process. As a result, it spends a lot of time wandering and exploring the workspace before finally arrives at the hole or even fails completely. As it gains more information about the structure of the workspace by actually performing the task, it gradually improves the performance by taking the highest valued action for each perceived state. This is achieved by reducing the value of ε towards 0. Consequently, the insertion time falls below 100 time steps after 8000 trials. Thereafter, this simulated control algorithm performs consistently in completing the insertion task within 100 time steps in the remaining training runs. Fig. 12 shows the rate of successful insertion over 100 consecutive training runs. The successful rate approaches 100% after 8000 trials and stays above 95%, thereafter. Since the peg is randomly placed at the start of each trial, this result indicates that the peg-in-hole task with positional uncertainties can be solved by TD-AVQ.

A control experiment is performed to compare the relative performance between fixed grid and AVQ for state space partitioning. The results show that the controller cannot achieve a single successful insertion in 10,000 trials. The reason is simply because the state

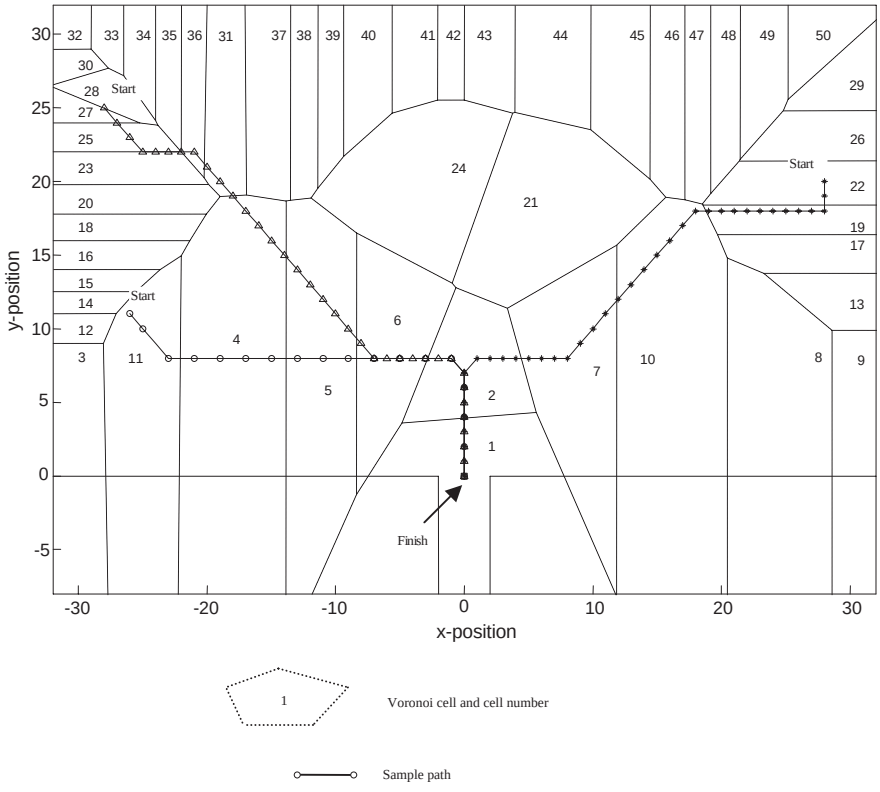

Fig. 10. The position space partition of the peg-in-hole task after 10,000 trials. Three samples paths from different starting positions are shown.

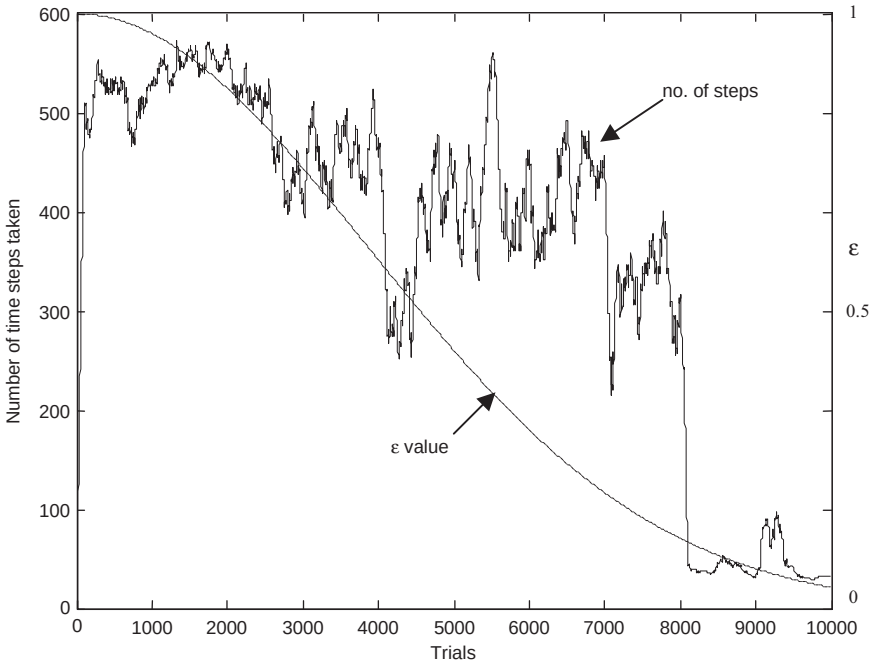

Fig. 11. The 200-point moving average of the number of time steps taken to complete the peg-in-hole insertion task. The maximum allowable number of steps for each trial is 600.

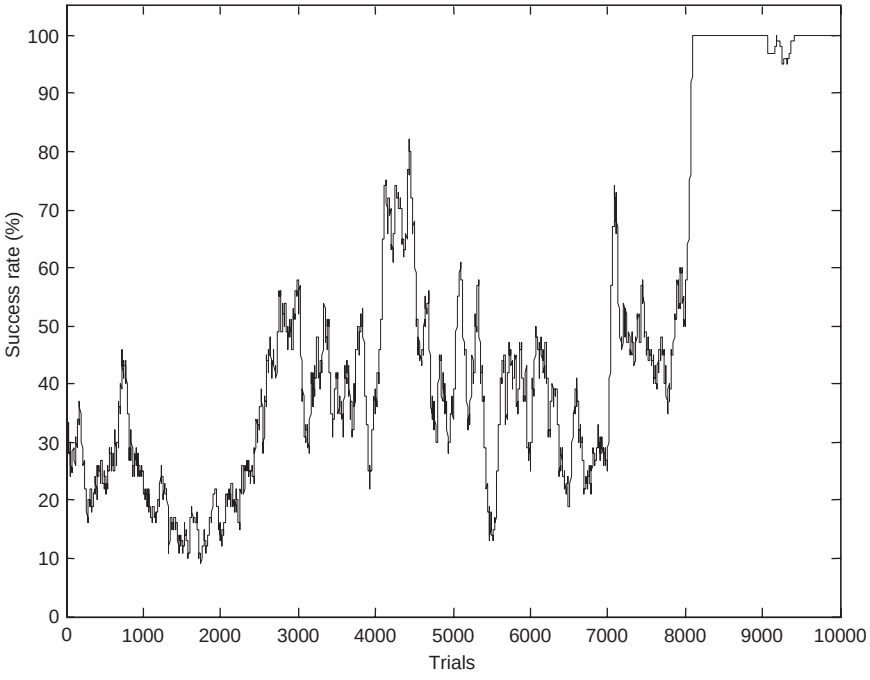

Fig. 12. The rate of successful insertion in 100 trials for the peg-in-hole task.

space is too large. With fixed grid partitioning, there are more than 19,000 distinguishable states. Adding to the fact that there are 8 admissible actions to choose from in each state, the chances of reaching the hole is very slim.

6. Conclusions

Reinforcement learning provides a framework for solving the decision problem faced by an autonomous agent working in an unknown environment. However,

the curse of dimensionality places a serious constraint on its applicability to large state space problems. The excessive computational workload hinders the applications of RL methods to real-time control tasks.

With a view to improve the applicability of RL, a novel adaptive partitioning scheme, TD-AVQ, that reduces the effective size of the state space for RL problems is developed. The properties of the proposed scheme are analyzed through a series of simulated navigation and peg-in-hole tasks. Empirical results show that on one hand the TD-AVQ algorithm reduces the number of distinguishable regions in the state space, and on the other hand is capable of solving the task. In particular, for tasks with critical regions such as the maze navigation and the peg-in-hole tasks, the number of states is reduced by 93% compared to using uniform grid partitioning. This causes a reduction in the number of decision stages (K), which, in turn, leads to an exponential decrease in the size of the control policy space. As a result, the computational complexity is reduced by 98.5% compared to uniform grid partitioning.

The simulation results for the peg-in-hole task suggest that a variable resolution method such as AVQ is essential for the successful application of reinforcement learning to problems with critical regions in the state space. With uniform grid partitioning, the model of the environment must be known in advance and the required minimum resolution for the critical region gives rise to an overly refined partitioning for the entire state space. On the other hand, TD-AVQ does not require any a priori knowledge of the environment. This partitioning scheme utilizes the information gained from the evaluation of action value function. To conclude, TD-AVQ improves the applicability of RL algorithms by reducing the overall computational complexity.

References

Bellman, R.E., 1957. Dynamic Programming. Princeton University Press, Princeton.

Bertsekas, D.P., Tsitsiklis, J.N., 1996. Neuro-Dynamic Programming. Athena Scientific, Belmont, Massachusetts.

Dayan, P., 1992. The convergence of TD(λ) for general λ. Machine Learning 8, 341–362.

Dayan, P., Sejnowski, T.J., 1994. TD(λ) converges with probability 1. Machine Learning 14, 295–301.

Gray, R.M., 1984. Vector quantization. IEEE ASSP Magazine 1 (2), 4–29.

Jaakkola, T., Jordan, M.I., Singh, S.P., 1994. On the convergence of stochastic iterative dynamic programming algorithms. Neural Computation 6, 1185–1201.

Kaelbling, L.P., Littman, M.L., Moore, A.W., 1996. Reinforcement learning: a survey. Journal of Artificial Intelligence Research 4, 237–285.

Lau, H.Y.K., Lee, I.S.K., 2001. Assembly skill acquisition via reinforcement learning. Assembly Automation 21 (2), 136–142.

Lau, H.Y.K., Mak, K.L., Lee, I.S.K., 2002. Adaptive vector quantization for reinforcement learning. In: Proceedings of the 15th World Congress of International Federation of Automatic Control, Barcelona, Spain, July 21–26, 2002.

Lee, I.S.K., Lau, H.Y.K. 2000. Assembly skill acquisition and model extraction via reinforcement learning. In: Proceedings of the 6th International Conference on Manufacturing (PCM 2000), pp. 776–781.

Mahadevan, S., Marchalleck, N., Das, K.T., Gosavi, A., 1997. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In: Proceedings of the 14th International Conference on Machine Learning, pp. 202–210.

Nomadic Technologies, Inc., 1997. Nomad 200 Hardware Manual.

Riedmiller, M., 1996. Application of sequential reinforcement learning to control dynamic systems. In: Proceedings of 1996 IEEE International Conference on Neural Networks, pp. 167–172.

Sutton, R.S., 1988. Learning to predict by the method of temporal difference. Machine Learning 3, 9–44.

Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA.

Tesauro, G.J., 1992. Practical issues in temporal difference learning. Machine Learning 8, 257–277.

Tesauro, G.J., 1994. TD-Gammon, a self-teaching backgammon program, achieves master-level play. Neural Computation 6 (2), 215–219.

Thrun, S.B., 1992. The role of exploration in learning control. In: White, D.A., Sofge, D.A. (Eds.), Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches. Van Norstrand Reinhold, New York, pp. 527–559.

Tsitsiklis, J.N., 1994. Asynchronous stochastic approximation and Q-learning. Machine Learning 16, 185–202.

Watkins, C.J.C.H., Dayan, P., 1992. Q-learning. Machine Learning 8, 279–292.

Zhang, W., Dietterich, T.G., 1996. High-performance job-shop scheduling with a time-delay TD(λ) network. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (Eds.), Advances in Neural Information Processing Systems 8: Proceedings of the 1995 Conference, pp. 1024–1030.