ORIGINAL ARTICLE

Adaptive Reinforcement Learning in Box-Pushing Robots

Kao-Shing Hwang[1, 2] and Jin-Ling Lin[3, *]

[1]*Department of Electrical Engineering, National Sun Yat-sen University, Taiwan*

[2]*Department of Healthcare Administration and Medical Informatics, Kaohsiung Medical University, Taiwan*

[3]*Department of Information Management, Shih Hsin University, Taiwan*

(**Received** 8 August 2017; **Accepted** 19 September 2017; **Published online** 1 March 2018)

*Corresponding author: jllin@mail.shu.edu.tw

DOI: 10.5875/ausmt.v8i1.1551

Abstract: An adaptive state aggregation Q-Learning method, with the capability of multi-agent cooperation, is proposed to enhance the efficiency of reinforcement learning (RL) and is applied to box-pushing tasks for humanoid robots. First, a decision tree was applied to partition the state space according to temporary differences in reinforcement learning, so that a real valued action domain could be represented by a discrete space. Furthermore, adaptive state Q-Learning, which is the modification of estimating Q-value by tabular or function approximation, is proposed to demonstrate the efficiency of reinforcement learning in simulations of a humanoid robot pushing a box. The box moves in the direction in which the robot asserts force. To push the box to the target point, the robot needs to learn how to adjust angles, avoid obstacles, and keep balance. Simulation results show the proposed method outperforms Q-Learning without using adaptive states.

Keywords: Adaptive state partition; Q-Learning; Reinforcement Learning; Humanoid robots; Cooperative box pushing

Introduction

Due to its unsupervised learning structure and ability to produce continual learning in a dynamic operating environment, reinforcement learning [1] has emerged as an important approach to machine learning. It allows robots to learn faster in searching for optimal strategies, if there is an effective way to cut the state space. However, it is hard to define a continual state space in reinforcement learning as a discrete state space and still get a robot to act as expected. Therefore, how to accommodate a continuously receptive field has become an important and complicated issue for research on how robots can best learn strategies via reinforcement learning.

Applying reinforcement learning in real environments requires state partition from the continuous information received and filed, because doing so significantly affects learning performance [2]. Some researchers have used neural networks as function approximators to estimate the state values [3][5]. Although they can avoid the state discretization problems, this approach has an important drawback in convergence, which is especially serious given complex state value functions. Tile coding uses a linear combination to evaluate the state value functions [6][8]. Although it can tackle continuous state and real value actions and uses less memory than directly partitioning the state space, choosing the tiling number and the number of tiles in every tiling instance is similar to choosing the partition sizes. Cognitive models based on adaptive resonance theory (ART) can adaptively partition the state space [9]. Such models require a certain amount of computational time to evaluate the vigilance parameter of all nodes; especially if the number of nodes is large. The decision trees can adaptively partition the state space into some exclusive subspaces corresponding to the leaf nodes [12]. Since robots only activate one node at every time step, they cannot easily combine actions chosen by activated nodes. They must also address issues related to the continuous action space.

Many existing reinforcement learning (RL) algorithms have been characterized with discrete sets of states and actions and can be transferred directly to real-world physical systems, general referred to as continuous state action spaces. Extensive research has recently focused on the development of new algorithms that can learn optimal policies to control physical systems described as continuous state and action spaces. Such approaches transfer these continuous variables to a new discrete version of the problem [15][16]. However, bad discretization of the state or action space may introduce hidden states into the problem, thus making it impossible to learn the optimal policy. On the other hand, too granular a discretization could moot the ability to generalize while increasing training data requirements. This is especially true when the task's state is multi-dimensional, where the number of discrete states can be exponential in the state dimension [17].

This paper proposes an adaptive state aggregation Q-Learning, such that humanoid robots can efficiently learn better strategies to reach their goal either in an individual or cooperative task in a dynamic environment. Humanoid robots were tasked to push boxes using dynamic parameter settings, and this activity was used to validate the practicality and effectiveness of the proposed learning method. The remainder of this paper is organized as follows. The research background is introduced briefly in the following section. Section 3 discusses the reinforcement learning method developed for box-pushing humanoid robots. Section 4 demonstrates the outcomes of box pushing simulations using a single and two cooperating robots with the proposed reinforcement learning method. Conclusions are drawn in the final section.

Kao-Shing Hwang (M'93-SM'09) is currently a professor of Electrical Engineering Department at National Sun Yat-sen University, and an adjoin professor of the department of Healthcare Administration and Medical Informatic, Kaohsiung Medical University, Taiwan. He received the M.M.E. and Ph.D. degrees in Electrical and Computer Engineering from Northwestern University, Evanston, IL, U.S.A., in 1989 and 1993, respectively. He had been with National Chung Cheng University in Taiwan from 1993-2011. He was the deputy director of Computer Center (1998-1999), the chairman of the Electrical Engineering Department (2003~2006), and the director of the Opti-mechatronics Institute of the university (2010~2011). He has been a member of IEEE since 1993 and a Fellow of the Institution of Engineering and Technology (FIET). His research interest includes methodologies and analysis for various intelligent robot systems, machine learning, embedded system design, and ASIC design for robotic applications.

Jin-Ling Lin received the master and Ph.D. degrees in computer science from the University of Oklahoma, Norman, OK, USA, in 1989 and 1993, respectively. She is currently a professor with the Department of Information Management, Shih Hsin University, Taipei, Taiwan. Her current research interests include machine learning, data science, data mining, information retrieval, intelligent system, intelligent network routing, and path planning in logistics.

Background

Q-Learning is a reinforcement learning technique, allowing machines to learn from actions without requiring a model of the environment. Reinforcement learning uses an action-value function to performs a given action *a* in a given state *s*. As shown in Fig. 1, the Q-Learning model can be described as the interaction of an agent, which can perform actions and alter the environment from one state to another, and its environment. The agent's memory contains a pair of state–actions and a *Q* value, computed from the action value function. The agent uses an evaluation function to select an action under state *s*, which has the best *Q* value, using the learning function to update the *Q* value during the learning procedure, and the reinforcement function to evaluate its action in the current state.

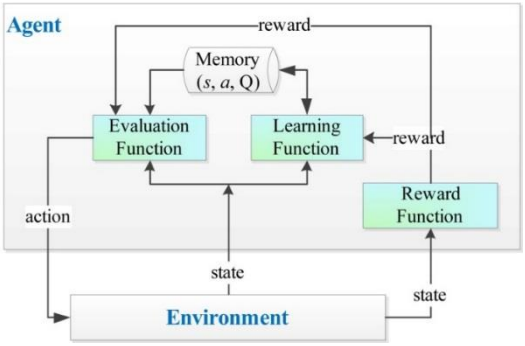


Figure 1. The Q-Learning model.

Before learning starts, *Q* is set to an arbitrary value. An agent performs an action *a* based on the state-action pairs stored in its memory under a specific state *s*. Then the state in the environment changes to *s'* and the agent obtains a reward for its action. The learning function computes a new *Q* value immediately after the agent performing an action. During training, the Q-Learning model uses these recorded *Q* values and online rewards to update the *Q* values in the agent's memory. After appropriate training, the *Q* value will converge to an optimal value. The *Q* value with state s and action *a* are updated as follows:

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma \max\left(Q(s',a')\right) - Q(s,a)\right] \quad (1)$$

where α, which is > 0 and $\leqq$ 1, is a real number used as the learning rate. γ, which is > 0 and $\leqq$ 1, is the temporal discount factor. *r* is the reward after performing action *a* under state *s* and can be evaluated according to different applications. *s'* denotes the state after performing action *a* under state *s*. max(*Q(s', a')*) denotes the maximum future *Q* value.

Reinforcement Learning for Box-Pushing Humanoid Robots

An adaptive state aggregation Q-Learning method is proposed to enhance the efficiency of reinforcement learning (RL) as applied to box-pushing tasks for humanoid robots. First, the decision tree was applied to partition the continuous state space according to temporary differences of reinforcement learning, so that a real value domain can be represented by a discrete space. Adaptive state Q-Learning was also applied. Figure 2 shows the proposed reinforcement learning for box-pushing humanoid robots. The upper part depicts the procedure of building decision trees based on the adaptive state aggregation (ASA-DT) approach, followed by the Q-Learning (ASA-QL).

Figure 2 shows the ASA-DT block 2, in which *unVisitedQueue* is a first in first out linear queue and was used to store these leaves for possible splitting. *Unvisited* represents the current leaf, which was tested for splitting. *Enqueue()* and *dequeue()* are functions respectively used to add a leaf for testing split into and remove a leaf from *unVisitedQueue*. The method of testing if a leaf needs to be split, choosing the splitting dimensions, and updating the state error are discussed in Section 3.1. When no leaf needs to be tested for splitting, the decision tree is regarded as being stable in the current environment. Humanoid robot box pushing with adaptive state aggregation is described in Section 3.2.

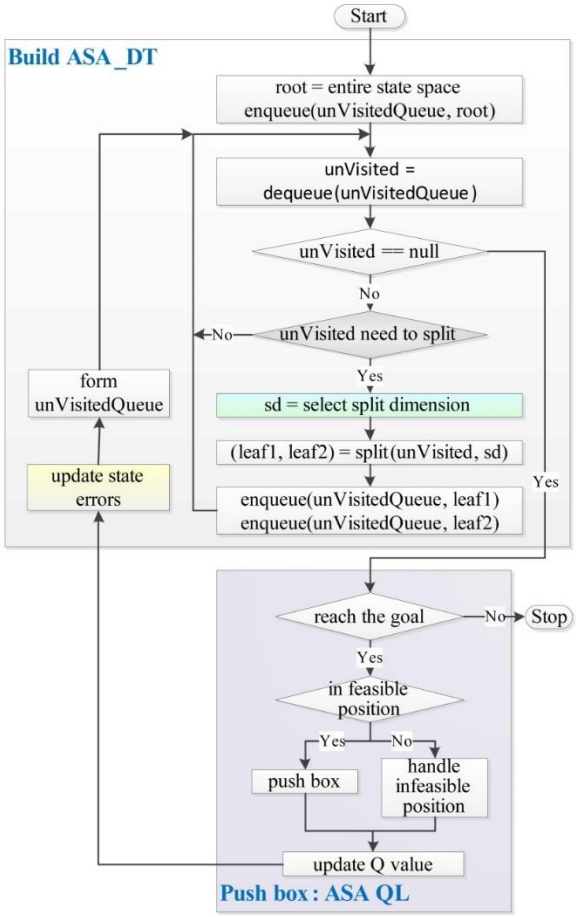


Figure 2. Reinforcement learning for box-pushing for humanoid robots.

Decision Tree for Adaptive State Aggregation (ASA-DT)

Discretization requires a continuous state space, and may categorize similar states into a single state. Reversely, a continuous action space can be interpolated from a set of some key or representative discrete actions. The proposed adaptive state aggregation method produces a decision tree that can adaptively segment the state space without expert experience. In the beginning, the decision tree only has a root node to represent the entire state space, including all information on sensory inputs and their estimation state errors. After several training iterations, the root node or some leaf nodes can be split into two child nodes analogous to two subspaces. When the training process is performed repeatedly, the tree grows and the state space is divided into subtler subspaces. Each internal node of the tree corresponds to a subspace in the state space. It can make a decision in the internal node and its decision represents a split in the subspace by means of a hyper plane perpendicular to the chosen split dimension. Each leaf node of the tree corresponds to an exclusive subspace of the state space and can be regarded as representing a state of the proposed adaptive state aggregation Q-Learning.

In a training iteration, the agent receives a sensory input vector u_t, and finds the most fitting leaf node, denoted as s, to cover u_t. Then, an agent selects an action according to the corresponding action values in the leaf node. After taking the action, the agent receives the next sensory input vector u_{t+1} and a reinforcement signal or reward r_{t+1}. The same action is repeated until the agent enters another leaf node s' if under semi-MDP [18]. The sojourning period, in which the agent keeps repeatedly performs the action found in the same node, is called an epoch and denoted as τ. The action value of each sensory input vector in the epoch at time $t+k$, where $0 \leqq k \leqq \tau$-1, can be expressed as follows [18]:

$$Q(\mathrm{u}_{t+k}, a) = r_{t+k} + \gamma r_{t+k+1} + \cdots + \gamma^{\tau-k-1} r_{t+\tau-1} + \gamma^{\tau-k} \overline{V}(s'), \qquad (2)$$

where γ, which is > 0 and $\leqq$ 1, is the temporal discount factor. The r is the reward and can be evaluated according to different applications. The $\overline{V}(s')$, defined as $\max_{a \in leaf\ s'} \overline{Q_{s'}}(s', a)$, is the estimated optimal state value in leaf node s'. The estimated action value of the leaf node s' is updated using the following formula:

$$\overline{Q}_s(s,a) = (1-\alpha)\overline{Q}_s(s,a) + \alpha[Q(u_t,a) + Q(u_{t+1},a) + \cdots + Q(u_{t+\tau-1},a)]/\tau \quad (3)$$

A state error, (u_{t+k}, err_{t+k}), is defined as the pairing of a sensory input vector u_{t+k} and its estimation error, where $err_{t+k} = Q(u_{t+k,a}) - \overline{Q_{s_t}}(s_t,a)$ is derived from Eqs. (2) and (3). Before updating the estimated state-action value of leaf node *s*, all estimation errors of these sensory input vectors and their corresponding state errors are recorded into a list of leaf nodes. After updating the estimated state-action value, the leaf node decides if a split is needed or not according to the distribution of sensory inputs and the magnitudes of estimation errors of the leaf node. If the number of records, pairs of (u_{t+k}, err_{t+k}) in leaf node *s*, is smaller than a predefined threshold, L_{th}, it implies that the estimated state-action values may not be convergent to a certain degree. Leaf splitting is only considered when the number of records about state errors is larger than L_{th}. When the number of records for state errors, N_{err}, is larger than L_{th}, the mean μ_{err} and variance σ^2_{err} of the estimation errors recorded in the leaf node is used to decide whether the leaf node should split or not. If the magnitude of the mean $|\mu_{err}|$ is large based on the size of state space, it means the learning process is still under progress and the leaf node cannot be split, otherwise the variance σ^2_{err} is used to determine the accuracy of the current estimation. When $|\mu_{err}|$ is close to zero, a small variance implies most errors are also small, thus the action values have been estimated accurately and the leaf node does not need to split. On the other hand, a large variance implies errors so divergent that a split may be essential for estimation improvement. In other words, if the mean of the estimation error is small, but its variance is large after the state-action values are trained and updated several times, it means that the leaf node cannot represent the subspace well and should be split into two child nodes [17]. The procedure of determining whether or not to split a leaf node is shown in Figure 3. Conditions for whether a leaf node *s* should be split are summarized as following:

Condition 1: The number of records for state errors, N_{err}, in leaf node *s* is larger than a predefined threshold L_{th}

Condition 2: The mean of estimation errors, $|\mu_{err}|$, in leaf node *s* is smaller than a predefined threshold μ_{th}, which is no less than 0

Condition 3: The variance of estimation errors, σ^2_{err}, in leaf node *s* is larger than a predefined threshold σ^2_{th}

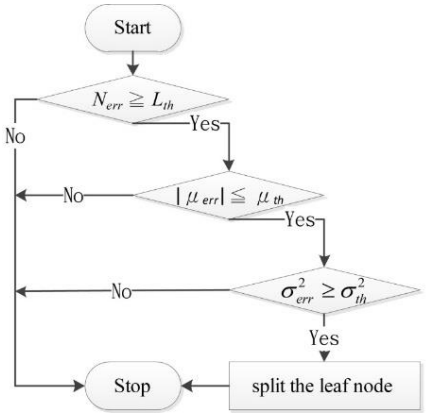


Figure 3. Determining whether leaf node *s* needs to split or not.

The weighted T statistic is used to select the split dimension [17]. The recorded state errors in a leaf node are divided into two groups; one has positive errors and the other has negative errors, and are respectively called g_p and g_n. In each group, elements of sensory input vectors in the same dimension are regarded as a set of statistic samples. The input vector is written as $X_{g,d} \sim (\mu_{g,d}, \sigma^2_{g,d})$, where $\mu_{g,d}$ and $\sigma^2_{g,d}$ are respectively the mean and the variance of dimension *d* of group *g*. If $X_{g_p,d} \sim (\mu_{g_p,d}, \sigma^2_{g_p,d})$ and $X_{g_n,d} \sim (\mu_{g_n,d}, \sigma^2_{g_n,d})$ have similar means and variances, it means the dimension *i* in the state variable has less influence on the estimation error. The T statistic is used to test the similarity of $X_{g_p,d} \sim (\mu_{g_p,d}, \sigma^2_{g_p,d})$ and $X_{g_n,d} \sim (\mu_{g_n,d}, \sigma^2_{g_n,d})$. The *t* value of the T statistic is calculated by Eq. (4).

$$t_d = \frac{\left|\mu_{g_p,d} - \mu_{g_n,d}\right|}{\sqrt{\frac{\sigma^2_{g_p,d}}{n_{g_p}} + \frac{\sigma^2_{g_n,d}}{n_{g_n}}}} \quad (4)$$

where n_{g_p} and n_{g_n} are respectively the number of elements in group g_p and g_n. A higher *t* value implies less similarity between groups g_p and g_n. The dimension with the highest *t* value is chosen as the split dimension. If all the state error records belong to one group only, the dimension with the highest variance is chosen as the split dimension. To avoid long training times, these two new leaf nodes inherit the state value from their parent node. The procedure for finding split dimensions is shown in Figure 4.

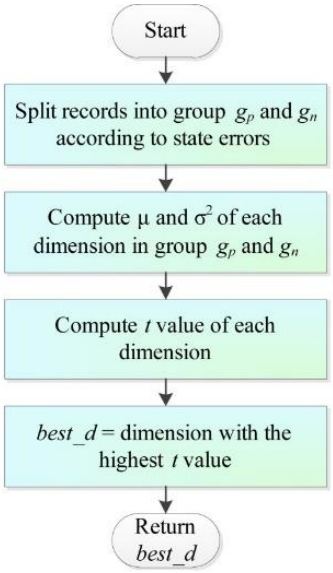


Figure 4. Finding split dimensions in a leaf node.

Adaptive Sate Aggregation Q-Learning (ASA-QL) for Box-Pushing Humanoid Robots

To push a box in an enclosed area, a humanoid robot requires seven representative actions: Turn Left, Turn Right, Stand Up, Push Up, Push Down, Push Left, and Push Right. The continuous state space is composed of the robots' coordinates, the box, obstacles, and the target area. A scenario example of humanoid robots pushing boxes is shown in Figure 5. The black box in front of the robot is the target box to be pushed, the narrow black box represents an obstacle, and the blue square denotes the goal.

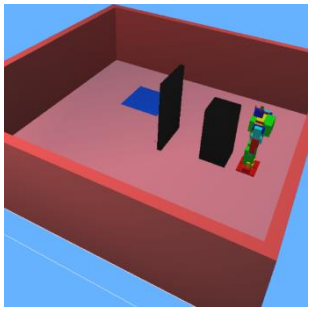


Figure 5. Humanoid robot box-pushing scenario.

Initially, the root node of the decision tree corresponds to the entire state space, which is represented by the coordinates of the robots and box to be pushed. After performing adaptive state aggregation reinforcement learning, internal nodes of the tree keep splitting until no remaining nodes need to be split. At the end of the training, an adaptive state aggregation decision tree is built, where each leaf node of the adaptive state aggregation decision tree corresponds to an exclusive subspace of the state space and can be regarded representing a state of Q-Learning.

Since the range of each state differs, the sojourn time, called an epoch, of each robot in each state is also different. When a robot enters a state, it is at a decision point and selects an action at the beginning of an epoch. It then maintains the action and observes the sensory inputs from the other observation points until the end of the epoch. In an epoch, the robot transited from one observation point to another many times, but all the observed sensory input vectors belong to the same representative state, akin to a leaf node in a decision tree. Therefore, the transition time of every epoch could be different in different environments.

For different environments, the robot continuously receives environmental information, including the position of the box, the goal, and the obstacles; responses to an action, and rewards or punishments obtained–depending on its surrounding environment. The robot controls the direction in which the box moves by applying force on the box's pushing point, thus the robot needs to learn how to adjust angles, avoid obstacles, keep its balance, and push the box to the goal. In other words, robots not only learn how to push the box to their goal, but also to maintain a consistent posture when pushing the box. For the robots to maintain gestures steadily while pushing the box, it must be able to stand up and learn how to adjust its angle of view to avoid surrounding walls after a fall. The robot must also learn other collision avoidance actions, such as avoiding collisions with static or moving obstacles. The robot learning procedure continues until the robot has pushed the box to the goal.

In addition, the ASA-QL can extend to an *n*-agent Markov decision process; that is, robots in a group can perform the ASA-QL simultaneously by extending the dimensions of the state space consisting of its own state space and the actions taken by its partners. Each robot has an associated reward function, described by a set of rules for all robots to use in the task.

Each robot's selected actions, next states and rewards depend on the joint actions of other robots. Therefore, the robot attempts to maximize its expected sum of discounted rewards. In practical terms, we assume that each robot does not know the reward functions and actions, but can only observe their immediate rewards. The evaluation of cooperative rewards relies on the Nash equilibrium, which requires each robot's policy be the best response to the others' policy. Although robots do not know other robots' reward functions, they can observe other robots' actions and subsequent rewards such that each robot performs an updated *Q* value whenever it receives a cooperative

reward when making a transition from *s* to *s'* after taking action *a*. The expected value in state *s* for robot *i* can be evaluated using the TD method as follows:

$$\overline{Q}_s(s,a^i,a^j) = (1-\alpha)\overline{Q}_s(s,a^i,a^j) + \alpha[Q(u_t,a^i,a^j) + Q(u_{t+1},a^i,a^j) + \cdots + Q(u_{t+\tau-1},a^i,a^j)]/\tau \quad (5)$$

where $\alpha \in [0,1]$ is the learning rate. The extended version is also applied to a task in which two robots learn to cooperatively push a box to a target area.

Simulation and Discussion

To evaluate the performance and practicality of reinforcement learning for box-pushing humanoid robots, simulations were conducted on a 3D mobile robot simulator (Webot 5.2.0). The Webot simulator can set environmental parameters (e.g., gravity and friction coefficient parameters) to make the proposed adaptive state aggregation Q-Learning method simulate a more realistic environment. It also can simulate different kinds of robots, e.g., wheeled, legged, and flying robots. These features allow users to design scenarios to reflect their actual applications. It also supports a range of programming languages, including C, C++, and JAVA. Therefore, once the training is finished, this well-trained box-pushing data can be applied to real world applications.

Figure 6 shows the simulation system, including a supervisor, robots, a box, and the environment in which the box and robots were located. Once the environment is set, the supervisor transmits the target box and an action to the robot. Depending on the robot's performance of the action, it is either rewarded or punished and sent it back to the supervisor for further learning.

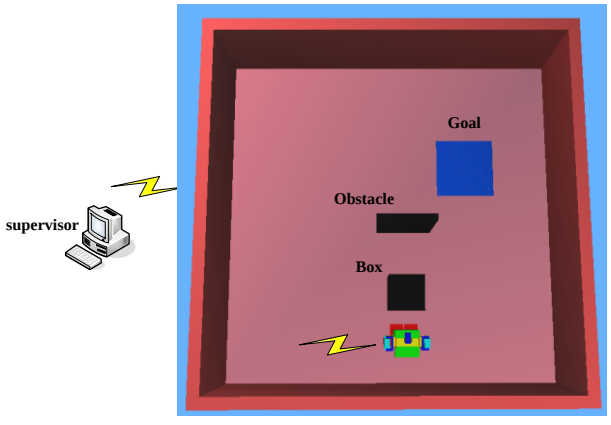
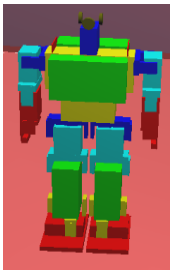


Figure 6. Simulation system.

The simulated environment was a field measuring 2 m * 2 m, while the goal was a 0.27 m * 0.27 m square, the box size was a cube measuring 0.15 m * 0.15 m * 0.3 m, and the obstacle measured 0.25 m * 0.25 m * 0.4. With each step, the robot moves 0.074 m. Each robot was equipped with three distance sensors (front, left, and right) to detect obstacles. The surveillance system comprises a wide-angle camera mounted on top of the robot to determine location coordinates, and a wireless antenna/receiver to maintain communications with the virtual supervisor and the other robots. As shown in Fig. 7, the humanoid robot consisted of 17 motors – head, left/right should, left/right arm, left/right hand, left/right hip, left/right leg1, left/right leg2, left/right ankle, and left/right foot. Each robot had seven predefined actions: turn left, turn right, stand up, push up, push down, push left, and push right.



(a) Simulation l robot

(b) Real world robot

Figure 7. Robot prototypes.

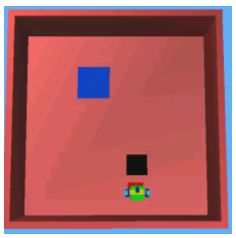
The simulated field was split into a 16*16 grid map. States were represented by robot_x, robot_y, box_x, and box_y, so the number of possible initial states was 16^4. The reward was set by the following rules:

- If the box hits the wall or an obstacle, deduct 100 points.
- If the robot pushed the box to the goal, award 100 points.
- If the Euclidean distance between the robot and the box exceeds as given threshold, deduct 30 points.
- For other statuses, deduct 1 point.

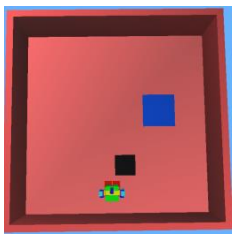
Four cases were simulated, as shown in Figure 8. The first three figures show a single robot pushing a box to the goal, with different initial positions for the robot with or without obstacles. The last image shows two robots cooperatively pushing a box to the goal with obstacles based on the method proposed in Section 3B. A single robot should push the box to the goal by itself. Two robots should cooperate to push the box to the goal, with *Q* values simultaneously updated using Eq. (5), but learning takes place individually. Each case has ten different initial positions for the robots and the box. The learning rate α was set to 0.8 and the discount rate γ was set to 0.9.

Figure 9 and Figure 10 show a single robot pushing a box positioned in front of the robot, to the goal located to the robot's left hand side, with and without obstacles. As shown in Fig. 9, after 100 training iterations, the robot took 23 steps to reach the goal without an obstacle. Figure 10(b) shows that the robot could choose to go to the left or the right of the obstacle, but selected the

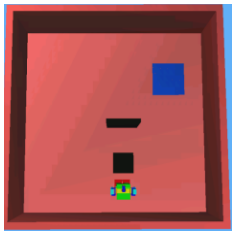
optimal (right) route, as, shown in Figure 10(c), taking 21 steps to reach the goal after 100 training iterations. In Fig. 11, two robots were positioned at initiation and tasked with jointly pushing the box to the goal. Both robots learned continuously and simultaneously but individually. Figure 11 shows the simulation of two robots pushing a box with an obstacle.



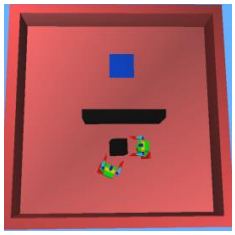
(a) A single robot pushing a box to the goal without obstacles. The box was in front of the robot, the goal was on the left hand side of the robot



(b) A single robot pushing a box to the goal without obstacles. The box was on the upper right side of the robot, and the goal was on the right side of the robot



(c) A single robot pushing a box to the goal with an obstacle

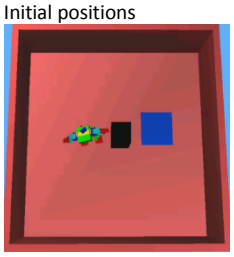


(d) Two robots pushing a box cooperatively

Figure 8. Four simulation cases.

(a) Initial positions

(b) After 8 steps



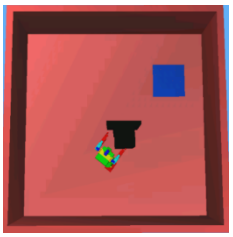
(c) After 20 steps

(d) After 23 steps, it reached the goal

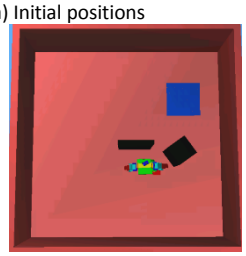
Figure 9. A single robot pushing a box without obstacles.

Finally, the proposed method was compared with Q-Learning without using adaptive states to show learning efficiency. Figure 12 compares the steps needed by the proposed method and conventional Q-Learning for a single robot to successfully push a box to the specified goal. Each tic of the horizontal axis represents ten trials (learning), the vertical axis, in an exponential presentation, shows the average number of steps for every 10 trials that the robot took to push the box to the goal. The purple/diamond line denotes the outcomes of conventional Q-Learning, which uniformly partitions the continuous state space into a set of regions and uses a table to record the experiences. The blue/X line indicates the results reached by the proposed method. The robot using the proposed method took fewer steps than those using conventional Q-Learning and the proposed method nearly converged after 150 trials.

(a) Initial positions



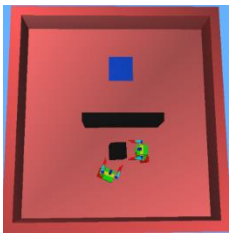
(b) Colliding with the obstacle



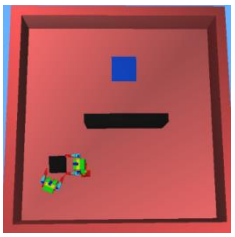
(c) Avoiding the obstacle

(d) After 21 steps, it reached the goal

Figure 10. A single robot pushing a box with an obstacle.



(a) Colliding with the obstacle



(b) Avoiding the obstacle

Figure 11. Two robots pushing a box with an obstacle.

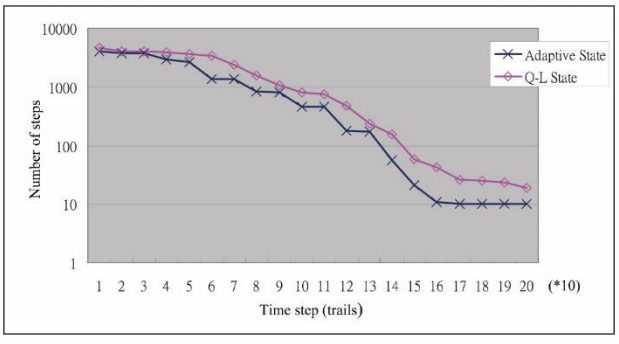


Figure 12. Performance comparison of Q-Learning with and without adaptive state aggregation.

Furthermore, Figure 13 shows the training data for the two robots cooperating to push a box. Each robot was trained individually. One dimension of the state space was expanded to accommodate actions the

partner might have taken. The *Q* values acquired in individual learning are "swept" along the new dimension to decrease the elapsed time in exploration. More than 500 steps are needed for the cooperating robots to reach the goal before the 100th trial (learning). During the learning trial, the average number of steps decreased to about 200 steps. After the 450th trial, the robots seem to have found an optimal solution and took about 10 steps to push the box to the goal.

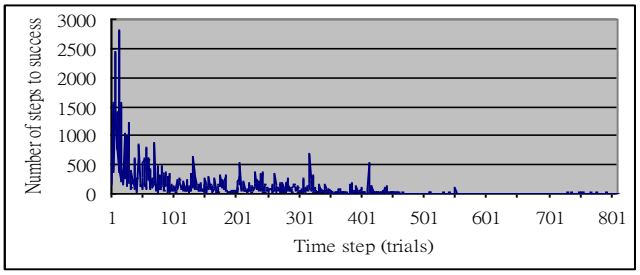


Figure 13. ASA-QL learning procedure for multiple robots jointly pushing a box.

Conclusions

An adaptive state aggregation Q-Learning (ASA-QL) is proposed for humanoid robots pushing a box. An adaptive data-based aggregation scheme to adaptively discretize a continuous state space was developed. The sate space is separated into regions of various sizes depending on the occurrences of states interacting with the environment. Since learning performance improved over time, learning results might not be correct in early episodes. Incorrect estimations of action values resulted in some redundant splits in tree growth. The proposed algorithm not only reduced redundant splits resulting from incorrect learning, but also pruned similar sibling nodes after each episode. It could clearly reduce the tremendous number of leaf nodes. To learn to accomplish a task, a growing decision tree as a way of building state space was first performed based on the proposed state partition method so that humanoid robots can learn more rapidly. Simulations had single or multiple robots learning to push a box with or without an obstacles. Comparing the number of steps needed to finish the box pushing task without and with ASA-QL showed the proposed ASA-QL outperformed the model without adaptive state aggregation.

Multi-agent systems differ from single-agent systems in that several agents exist in the environment modeling each other's goals and actions. From an individual agent's perspective, multi-agent systems vary from single-agent systems most significantly because the environment's dynamics can be affected by other agents. In addition to system uncertainty, other agents may intentionally affect the environment. In the case of multiple agents learning simultaneously, one particular agent is learning the value of actions in a non-stationary environment. Thus, the convergence of the aforementioned Q-Learning algorithm is not guaranteed in a multi-agent setting. Given certain assumptions about the way in which actions are selected at each state over time, Q-Learning converges to the optimal value function. The simplest way used here was just to add the partner's actions to extend the state space such that the learning agents can pretend that the environment is stationary. Unfortunately, this may work only in the cases shown in the simulations, and there is no guarantee for the successful completion of more complicated tasks or given more peers in a group.

References

[1] Wikipedia. Reinforcement Learning. 2012. http://en.wikipedia.org/wiki/Reinforcement_learning (accessed Aug. 05, 2017).

[2] L. D. Pyeatt and A. E. Howe, "Decision Tree Function Approximation in Reinforcement Learning," Technical Report CS-98-112, Fort Collins, Colorado, Colorado State University, 1998.

[3] B. Baddeley, "Reinforcement Learning in Continuous Time and Space: Interference and Not Ill Conditioning Is the Main Problem When Using Distributed Function Approximators," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 38, no. 4, pp. 950-956, 2008. doi: 10.1109/TSMCB.2008.921000

[4] S. J. Bradtke and A. G. Barto, "Linear Least-Squares Algorithms for Temporal Difference Learning," *Machine Learning* 22, pp. 33-57, 1996. doi: 10.1007/BF00114723

[5] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in proceeding of *Advances in Neural Information Processing Systems* 12, Nov. 29- Dec. 4. 1999, MA, USA, pp. 1057-1063.

[6] K.-S. Hwang, Y.-J. Chen, and T.-F. Lin, "Q-Learning in Multi-Agent Cooperation," in proceeding of *IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO 2008),* Taipei, Aug. 23-25, 2008, pp. 1-6. Doi: 10.1109/ARSO.2008.4653621

[7] S. Sutton, "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding," *Advances in Neural Information Processing System*, no. 8, pp. 1038-104, 1996.

[8] Y. Zheng, S. Luo, and Z. Lv. "Control Double Inverted Pendulum by Reinforcement Learning with Double CMAC Network," in proceeding of *18th International Conference on Pattern Recognition (ICPR 2006)*, pp. 639-642, 2006.
doi: 10.1109/ICPR.2006.416

[9] K.-S. Hwang, Y.-P. Hsu, H.-W. Hsieh, and H.-Y. Lin, "Hardware Implementation of FAST-Based Reinforcement Learning Algorithm," in proceeding of *2005 IEEE International Workshop on VLSI Design and Video Technology*, pp. 435-438, 2005.
doi: 10.1109/IWVDVT.2005.1504643

[10] A.-H. Tan, N.-Lu, and D. Xiao, "Integrating Temporal Difference Methods and Self-Organizing Neural Networks for Reinforcement Learning With Delayed Evaluative Feedback," *IEEE Transactioins on Neural Networks,* vol. 19, no. 2, pp. 230-244, 2008.
doi: 10.1109/TNN.2007.905839

[11] H. Ueda, T. Naraki, Y. Nasu, K. Takahashi, and T. Miyahara, "State Space Segmentation for Acquisition of Agent Behavior," in proceeding of *IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'06)*, Hong Kong, China, Dec. 18-22, 2006, pp. 440-446.
doi: 10.1109/IAT.2006.113

[12] D. Chapman and L. P. Kaelbling, "Input Generalization in Delayed Reinforcement Learning: An Algorithm And Performance Comparisons," in proceeding of *12th International Joint Conference on Artificial Intelligence (IJCAI-91),* Sydney, Australia, Aug. 24-30, 1991, pp. 726-731.

[13] K.-S. Hwang and Y.-J. Chen, "Tree-like Function Approximator in Reinforcement Learning," in proceeding of *33rd Annual Conference of the IEEE Industrial Electronics Society (IECON 2007)*, Nov. 5-8 2007,Taipei, pp. 904-907.
doi: 10.1109/IECON.2007.4460012

[14] R. Munos and A. Moore, "Variable Resolution Discretization in Optimal Control," *Machine Learning,* vol.49, no. 2-3, pp. 291-323, 2002.
doi: 10.1023/A:1017992615625

[15] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research,* vol. 4, pp. 237-285, 1996.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction,* Cambridge: MIT Press, 1998.

[17] Y.-J. Chen, “A Self-Organizing Decision Tree Approach to Policy Sharing of Multi-Agent Systems,” Ph.D. Dissertation, Department of Electrical Engineering, National Chung Cheng University, Taiwan, 2009.

[18] T. K. Das, A. Gosavi, S. Mahadevan, and N. Marchalleck, "Solving Semi-Markov Decision Problems Using Average Reward Reinforcement Learning," *Management Science,* vol. 45, no. 4, pp. 560-574, 1999.
doi: 10.1287/mnsc.45.4.560