Proceedings of the Eighth International Conference on Machine Learning and Cybernetics, Baoding, 12-15 July 2009

Finding Shortcuts from Episode in Multi-Agent Reinforcement Learning*

Zhao Jin[1], WeiYi Liu[1], Jian Jin[2]

[1]School of Information Science and Engineering, Yunnan University, Kunming, 650091, P.R.China
[2]Hongta Group Tobacco Limited Corporation, Hongta Road 118, Yuxi, 653100, P.R.China
E-MIAL: jzletter@sina.com, liuweiyi2000@yahoo.com.cn

Abstract

In multi-agent reinforcement learning, the state space grows exponentially in terms of the number of agents, which makes the training episode longer than before. It will take more time to make learning convergent. In order to improve the efficiency of the convergence, we propose an algorithm to find shortcuts from episode in multi-agent reinforcement learning to speed up convergence. The loops that indicate the ineffective paths in the episode are removed, but all the shortest state paths from each other state to the goal state within the original episode are kept, that means no loss of state space knowledge when remove these loops. So the length of episode is shortened to speed up the convergence. Since a large mount of episodes are included in learning process, the overall improvement accumulated from every episode's improvement will be considerable. The episode of multi-agent pursuit problem is used to illustrate the effectiveness of our algorithm. We believe this algorithm can be introduced into most other reinforcement learning approaches for speeding up convergence, because its improvement is made on episode, which is the most foundational learning unit of reinforcement learning.

Keywords:

multi-agent reinforcement learning, episode, state loops, shortcut, speed up convergence

1. Introduction

Reinforcement learning is a promising technique to solve the problem in an uncertain and dynamic multi-agent setting, because its trail-and-error mechanism is suitable the situation that each agent is difficult, even impossible to predict or plan other agents' actions in advance[1][2]. For example, in multi-agent pursuit problem each hunter can not predict actions of other hunters (without communication) and preys in advance. But the problem that the state space grows exponentially in terms of the number of agents becomes an obstacle to apply this technique to multi-agent domain [1][3].

Some approaches have been proposed to solve this increased exponentially state space problem. To reduce the number of states by coarse-graining of perception is proposed in[4]. A hybrid approach that combines modular-Q learning and profit-sharing learning for improving the convergence speed in large state space is presented in[5]. To decompose task in large state space into subtasks with small state space for speeding up learning is introduced in[6][7]. A cut-loop routine is introduced into profit-sharing learning to shorten the length of episode for speeding up convergence in[8]. We think to improve the convergence efficiency of episode is inspiring, because the learning process includes thousands of episodes even in small state space. The improvement for every episode during the learning process will make up of great improvement for the overall performance of learning.

But in Arai's approach[8], the way to cut off loops is problematic. For loop made up by two repeated states (the latter is the revisited one), all states except one of the two repeated states are deleted directly to remove loop. Some problems are raised from this way. These deleted states in the loops can not be refined, and the state paths from these deleted state to goal state within the original episode are also lost, that means losing the state space knowledge have been learned from original episode. To remove loops in this way, even the states kept still in the processed episode that remove the loops can be refined faster, but these deleted states will can not be refined. It seems like the agent did not reach these deleted states at all. In fact, the overall performance of learning is degraded.

*This work was supported by the National Natural Science Foundation of China (No. 60763007), the Chun-Hui Project of the Educational Department of China (No. Z2005-2-65003)

978-1-4244-3703-0/09/$25.00 ©2009 IEEE

In this work, we proposed an algorithm to find shortcuts from episode in multi-agent reinforcement learning for improving the efficiency of the convergence. By this algorithm, we remove all state loops indicate the ineffective paths and irrational agent's behavior in the episode, but keep all shortest state paths form each other state to the goal state within the original episode. That is, we not only speed up convergence by removing loops from episode, but also ensure every state in the original episode can be refined by goal state reward, that avoids the state space knowledge loss problem in[8]. In addition, the shortest state path from a state to goal state represents an effective and rational state-action control policy leading to the goal state. It is also desirable to provide an effective or rational solution, even not optimal, especially in uncertain and dynamic multi-agent setting[8][9].

Note that a large amount of episodes are included in the reinforcement learning process. So the overall improvement accumulated from every episode's improvement will be considerable. We believe the algorithm can be introduced into most other reinforcement learning approaches for speeding up convergence, because its improvement is made on episode, which is the most foundational learning unit.

The rest of this paper is organized as follows. Section 2 presents the algorithm to learn shortcuts from the episode and an example of the multi-agent pursuit problem. Sections 3 are conclusions and future works.

2. The algorithm to learn shortcuts form episode

2.1. related works

The state-action transition sequence that the agent moves from some initial state to the final reward state is known as an episode. The episode is the most foundational learning unit in reinforcement learning. A large amount of episodes are included in the learning process, thus a slight improvement on episode will accumulate huge overall improvement on reinforcement learning. Except to be as the learning unit, the episode is also used in different ways. In Profit-sharing learning[10], the episode is recorded to assign credit to each state-action pair in it to learn effective behavior. Mitchell[11] proposed to store episodes for faster refining value function in Q-learning. Tan[12] used episodes as the communication media for state space knowledge exchange in multi-agent cooperative learning.

However, these application of episode do not take the loops in episode into consideration. These loops in episode exhibit irrational behavior with respect to achieving its goal. Although in general, the acquired policy need not be optimal for multi-agent setting, it is important that the policy is rational. A rational policy is one that is guaranteed to converge on a solution, i.e. the agent should not be trapped within infinite loops in the episode. Some researchers also noted this, Miyazaki[13] gave smaller rewards to the state-action pairs which make up the loop in the episode. Arai[8] introduced a simple but rough way to remove the loops, but this way makes the loss of state space knowledge, which cause to degrade the overall learning performance.

It is very valuable to provider a method to remove the loops in episode without loss of state space knowledge. One aspect it can at least provide an effective and rational solution for the learning problem; Another aspect it can significantly improve the efficiency of the convergence because a large amount of episodes are included in the learning process. Next we give an introduction of the multi-agent pursuit problem.

2.2. The multi-agent pursuit problem

The pursuit problem[14] is a typical problem in multi-agent setting, here is a slight variant of it. In a 9×9 grid world, a prey agent and four hunter agents are placed at random positions in this grid world. Hunters are learning agents and try to capture the randomly moving prey. At each time(step), agents select randomly one of four actions: $moveUp$, $moveRight$, $moveDown$ and $moveLeft$ to perform, without communicating with each other. The prey and the hunters can not share a grid, but more than one hunter can be at the same grid. When the fox is surrounded by the hunter from its entire neighbor grid, the fox are captured. An episode is shown in Figure 1. Some notations and illustration for this pursuit problem shown in Figure 1 are given as follows:

1. The two numbers on each grid's top left corner are the row and column of this grid, noted as rc, is used to mark the locations of agents, for example, the prey's location is 64 in Figure 1.
2. The grids with a circle in the center are the begin locations of agents, for example, the prey's begin location is 55 in Figure 1.
3. Each hunter is distinguished from other with the number on top right corner of this hunter, the four hunters are noted as $h1$,$h2$,$h3$ and $h4$.
4. Each agent's action in each step is shown as an arrow in different color, and the number neighbored arrow is the move order, for example, hunter $h3$ takes action $moveDown$ from location 83 to 93 on the third step.
5. The state is a vector composed of every agent's location, noted as $S_i(rc, rc, rc, rc, rc)$, where i is the serial number of state for distinguishing different states, the first state is S_0, when reach a new state, the serial number is added by 1. For convenient, in all this paper, we define the first rc is the location of $h1$, the second rc is the location of $h2$, the third

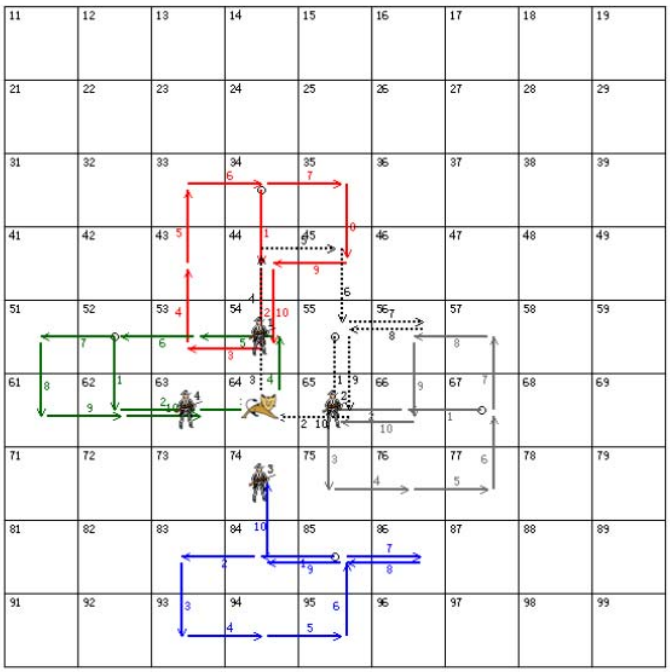

Figure 1. An episode in multi-agent pursuit problem

and fourth rc are separately the locations of $h3$ and $h4$, and the last rc is the location of prey. For example, the initial state is $S_0(34, 67, 85, 52, 55)$.

6. The action is also a vector composed of every agent's action, noted as A(a,a,a,a,a). We define the first a is the action of $h1$, the second a is the action of $h2$, the third and fourth a are separately the actions of $h3$ and $h4$, and the last a is the action of prey. For example, An Action $A(moveDown,\ moveLeft,\ moveLeft,\ moveDown,\ movedown)$ is taken to move theses agents from the state $S_0(34, 67, 85, 52, 55)$ to the state $S_1(44, 66, 84, 62, 65)$. In this problem, the joint action can be inferred from the current state and the next state, So we do not record the joint actions in the episode, only states are recorded in the episode.

From Figure 1, we have an episode as Equation (1):

$$\begin{aligned}
&S_0(34, 67, 85, 52, 55)(*)(initial\ state)\\
&S_1(44, 66, 84, 62, 65)(\wedge)\\
&S_2(54, 65, 83, 63, 64)\\
&S_3(53, 75, 93, 64, 54)\\
&S_4(43, 76, 94, 54, 44)\\
&S_5(33, 77, 95, 53, 45)\\
&S_0(34, 67, 85, 52, 55)(*)\\
&S_6(35, 57, 86, 51, 56)\\
&S_7(45, 56, 85, 61, 55)\\
&S_1(44, 66, 84, 62, 65)(\wedge)\\
&S_8(54, 65, 74, 63, 64)(goal\ state)
\end{aligned} \tag{1}$$

It is obvious that there are two interlaced loops(marked with $*$ and $\wedge$) in this episode. Next subsection we give the algorithm to find shortcuts form episode.

2.3. To learn shortcuts from episode

We rewrite Equation (1) as Equation (2) for convenient, the exact locations and joint actions of these agents can be checked from Equation (1) and Figure 1.

$$S_8 \leftarrow S_1 \leftarrow S_7 \leftarrow S_6 \leftarrow S_0 \leftarrow S_5 \leftarrow S_4 \leftarrow S_3 \leftarrow S_2 \leftarrow S_1 \leftarrow S_0 \tag{2}$$

If use the way in[8], it will be confused to select which loop to remove. To remove the loop made up by repeated state S_1, the result is:

$$S_8 \leftarrow S_1 \leftarrow S_0$$

To remove the loop made up by repeated state S_0, the result is:

$$S_8 \leftarrow S_1 \leftarrow S_7 \leftarrow S_6 \leftarrow S_0$$

No matter which loop is selected to remove, the state space knowledge are lost. These deleted states in the loop will can not be refined in this episode, which causes to degrade the overall performance of learning.

We propose a comprehensive way to remove loops and find shortcuts from episode for speeding up convergence. The basic idea is described as follows:

First, to create a null list sc to be the first shortcut, and sc is added a shortcuts set sc_set

Then, from the last state of episode begin, in reverse order, each state in episode is processed according to follow two situations:

1) if s dose not exist sc, then add s into sc.

2) if s has already existed in sc, and the index of s in sc is i. In this situation, it is obvious if we add s into sc again, a loop is made up by two repeated s. The sate s should be passed over.

It dose need to worry about the omitted state s can not be refined or refined with smaller reward that means to degrade the convergence. First, it has already existed in sc, so it can be refined; Second, in most reinforcement learning methods based value function iteration(e.g.,[8][15][16]),the state is closer to final reward state, it will be refined with bigger reward. So, even add s into sc again, it index position in sc is absolutely bigger than i, that is further from final reward state. Because it is the last element of sc. Therefore, to pass over s in this situation is actually beneficial to the convergence of state s.

But for keeping the state paths in the episode, a fork should be created from state s of sc to link these subsequent states after state s in the episode. For program implementation convenient, a new list, that is a new shortcut, with the elements of index from 0 to i of sc is created to be as the fork from state s to link the subsequent states after state s in episode. For example, When we processed the tenth state S_1 in Equation(2), we will get two lists shown in Equation (3). The second new list can be regarded as the fork stretched from state S_1 in the first list. We use it to link the subsequent state S_0 of the tenth state S_1 in Equation(2). After that, the new shortcut is added into sc_set.

$$\begin{array}{l} S_8 \leftarrow S_1 \leftarrow S_7 \leftarrow S_6 \leftarrow S_0 \leftarrow S_5 \leftarrow S_4 \leftarrow S_3 \leftarrow S_2 \\ S_8 \leftarrow S_1 \leftarrow (to\ link\ subsequent\ states) \end{array} \quad (3)$$

Now we keep the state path. But a new problem raises when we use the state s with the smaller index, that is closer to final reward state to link these subsequent states after state s in episode. The subsequent state may also has already existed in some list of sc_set, and the index in that list is bigger, that is further from final reward state than the index that it is added into the new list. For example, when we processed the last repeated state S_0 in Equation(2), we will get the result like Equation (4):

$$\begin{array}{l} S_8 \leftarrow S_1 \leftarrow S_7 \leftarrow S_6 \leftarrow S_0(far) \leftarrow S_5 \leftarrow S_4 \leftarrow S_3 \leftarrow S_2 \\ S_8 \leftarrow S_1 \leftarrow S_0(close) \end{array} \quad (4)$$

For faster convergence, in other word, for find the shortest state path from S_0 to final goal state, the sub list made up by state S_0 and its subsequent states in the first list should be transferred to the state S_0 with smaller index in the second list, that is closer to final reward state. The last result should be like Equation (5).

$$\begin{array}{l} S_8 \leftarrow S_1 \leftarrow S_7 \leftarrow S_6 \\ S_8 \leftarrow S_1 \leftarrow S_0 \leftarrow S_5 \leftarrow S_4 \leftarrow S_3 \leftarrow S_2 \end{array} \quad (5)$$

At last, we get the shortcuts found from episode.

A complete algorithm to find shortcuts from episode is given in Algorithm 1.

Algorithm 1. Find Shortcuts

Input

e: an episode in multi-agent reinforcement learning

Output

sc_set: a set of shortcuts, computed from e

Step 1. Initialization:

Create a null set: sc_set
Create a null list: sc, marked as current list
Add sc into sc_set
Let $i \leftarrow$ (the length of $e-1$)
Let s is a state, and $s \leftarrow e[i]$
Let $joint \leftarrow$ the length of sc

Step 2. While i ≥ 0 do

If s does not exist in any list of sc_set
Add s into current list sc
Else if s exist in some lists of sc_set
Let l^m is one of these list in sc_set who include s, and the index of s in l^m is minimum.
Let $index_{\min}$ equal to the index of s in l^m
If $joint > index_{\min}$
Create a new list: l^n
Copy these elements which index from 0 to $index_{\min}$ of l^m into l^n
Add l^n into sc_set
Let $sc \leftarrow l^n$
Else if $joint = index_{\min}$
Add s into sc
Else $joint < index_{\min}$
For each list l^s in sc_set that include s
Let $m \leftarrow$ the index of s in l^s
Create a new list: l^n
Copy these elements which index from 0 to ($joint$-1) of sc into l^n
Transfer these elements which index from m to the last one of l^s into l^n
Add l^n into sc_set
End for
Add s into sc
End if
End if
Let $joint \leftarrow$ the length of sc
Let $i \leftarrow i-1$
Let $s \leftarrow e[i]$
End While

End Algorithm 1.

We assume the multi-agent pursuit problem is computed by Profit-sharing learning[8][10]. The original episode shown in Equation (1) and the shortcuts found form it by Algorithm 1 shown in Equation (5) are used to compare the convergence efficiency of them. In Profit-sharing learning, the credit assignment function is defined as

$$f(R,t) = R \times \beta^{T-t} \tag{6}$$

where T is the total steps that the agent moves from some initial state to the final reward state in an episode, for example, T is equal to 11 in the episode of equation (1), and t is the count of steps that the agent move from some initial state to the current state, for example, when achieve $S3$ in equation (1), t is equal to 4. β is the discount rate, $0 < \beta < 1$, that means when the agent is further from the goal state, that is t is smaller, then a smaller credit is assigned to the agent's state-action pair at the tth step. R is the reward the agent is given when it achieves the goal state, and it is expressed as a numeric value. The reward R is used to compute a numeric credit with parameters β, T and t in equation (6) for each state-action pair in the episode to learn more effective behaviors(state-action pairs with bigger credit).

We only care about the credit value comparisons among state-action pairs not the actual credit value of each state-action pair, because the aim of learning is to find more effective state-action pair which have bigger credit value. In most situation the initial credit of each state-action pair is assigned zero, and it is reasonable because we initially do not know whether these state-action pairs can lead goal state. Although these parameters can affect the speed of learning, to give a best set of parameters for all situations is impossible. Our aim is to demonstrate the shortcuts found in episode can speed up convergence, so we simply adopt the reward R achieved final reward state S_8 is 100, and the discount rate β is 0.9, then the credit assigned each state-action pair by equation (6) in the Profit-sharing approach with original episode(abbreviated PS) and Profit-sharing approach with found shortcuts (abbreviated PSSC) are shown in Table 1 and Figure 2, note that only the state is listed, the action can be checked from Figure 1.

Table 1. A credit comparison of PS and PSSC

state	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
PS	35	39	43	48	53	59	73	81	100
PSSC	81	90	53	59	66	73	73	81	100

From Table 1 and Figure 2, the states in PSSC get larger credits than they are in PS, because we get the shortest state paths from each states to the goal state by finding shortcuts from original episode. Therefore, with the shortcuts,

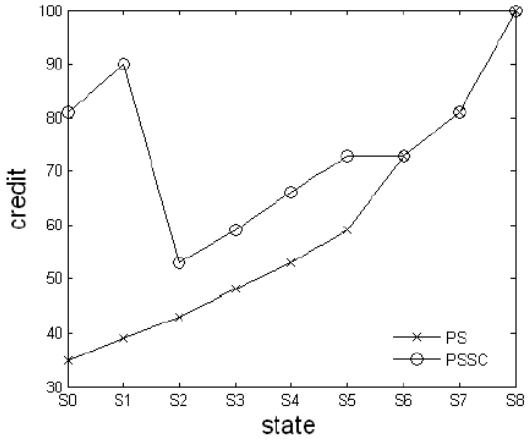

Figure 2. A credit comparison of PS and PSSC

the Profit-sharing approach will converge faster. Of course, this algorithm to find shortcuts from episode can also be applied to most other reinforcement learning approach, because episode is the most foundational learning unit of most reinforcement learning approaches.

This algorithm can be implemented easily to run on computer. For each episode recorded during learning process, all loops in it can be removed while all shortest state path from each state in this original episode to the final reward state are kept. When the episode is more and longer, the effect of this algorithm is more obvious. Considering large amount of episodes are included in learning process, the overall learning performance will be improved significantly.

3. Conclusions

The uncertainty and dynamic of problem domain is the source for appealing reinforcement learning, because its trail-and-error mechanism adapts to this situation. But the tremendous state space makes the episode is too long, and may exist a large amount of loops that exhibit irrational and ineffective behaviors of the agents and decrease the efficiency of the convergence.

We proposed an algorithm to learn shortcuts from episode in multi-agent setting for improving the efficiency of convergence. In this approach, the loops in episode are removed to speed up convergence, while keep all state space knowledge acquired form original episode, to ensure each state in the original episode can be refined by the reward acquired from achieving the final reward state. By this way, the episode is shortened to speed up convergence, and no any loss in state space knowledge. Since the improvement is made on every episode during the learning process, the ac-

cumulative improvement will be considerable. The example to use this approach to shorten the episode of pursuit problem demonstrates the effectiveness of this algorithm. We believe this algorithm can be introduced to most other reinforcement learning approaches, because the improvement is made on the most foundational learning unit: episode.

The future works can be focus on combining the knowledge acquired form each episode to accelerate convergence and merge this algorithm to other reinforcement learning approaches.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (No. 60763007), the Chun-Hui Project of the Educational Department of China (No. Z2005-2-65003).

References

[1] Y. Shoham, R. Powers, If multi-agent learning is the answer, what is the question? Artificial Intelligence, Vol.171, pp. 365-377, May 2007.

[2] L. Panait, S. Luke, Cooperative Multi-Agent Learning: The State of the Art, Autonomous Agents and Multi-Agent Systems, Vol.11, No.3, pp. 387-434, 2005.

[3] G. Tesauro, Extending Q-learning to general adaptive multi-agent systems, In Advances in Neural Information Processing Systems, Vol. 16, 2004.

[4] Akira Ito, Mitsuru Kanabuchi, Speeding Up Multiagent Reinforcement Learning by Coarse-Graining of perception: The hunter Game, Electronics and Communications in Japan, Part2, Vol.84, No.12, pp. 37-45, 2001.

[5] P.C. Zhou, B.R. Hong, Hybrid Multiagent Reinforcement Learning Approch: The pursuit Problem, Information Technology Journal, Vol.5 , No.6, pp. 1006-1011, 2006.

[6] N. Mehta, P. Tadepalli, Multi-agent shared hierarchy reinforcement Learning, Proceedings of the ICML05 Workshop on Richer Representations in Reinforcement Learning,Bonn,Germany, 2005.

[7] R. Makar, S. Mahadevan, M. Ghavamzadeh, Hierarchical multi-agent reinforcement learning, Autonomous Agents and Multi-Agent Systems, Vol.13, No.2, pp. 197-229, 2006.

[8] S. Arai, K.P. Sycara, R.P. Terry, Experience-based Reinforcement Learning to Acquire Effective Behavior in a Multi-agent Domain, Proceedings of the 6th Pacific Tim International Conference on Artificial Intelligence, pp. 125-135, 2000.

[9] C. Claus, C. Boutilier, The dynamics of reinforcement learning in cooperative multiagent systems, in: Proceedings of the Fifteenth National Conference on Artificial Intelligence, pp. 746-752, 1998.

[10] J. J. Grefenstette, Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms, Machine Learning Vol.3, pp. 225-245, 1988.

[11] T.M. Mitchell, reinforcement learning, Machine Learning, McGraw-Hill, New York, pp. 367-390, 1997.

[12] M. Tan, Multi-agent reinforcement learning: Independent vs. cooperative agents, in: Proceedings of the 10th International Conference on Machine Learning, Amherst, USA, pp. 330-337, 1993.

[13] K. Miyazaki, S. Kobayashi, On the Rationality of Profit Sharing in Partially Observable Markov Decision processes,Proceedings of the 5 thInternational Conference on Information systems analysis and synthesis, pp.190-197, 1999.

[14] M. Benda, V. Jagannathan, R. Dodhiawalla, On optimal cooperation of knowledge sources. Tech Rep BCSG2010-28, Boeing AI Center,1985.

[15] R. S. Sutton, Learning to predict by the methods of temporal difference,Machine Learning, No.3, pp. 9-44, 1988.

[16] C.J.Watkins, Learning from delayed rewards,PhD Thesis,University of Cambridge,England,1989.