Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

The Improvement of Q-learning Applied to Imperfect Information Game

Jing Lin[1], Xuan Wang[1], *Member, IEEE,* Lijiao Han[2], Jiajia Zhang[1], Xinxin Xu[1]
[1]Intelligence Computing Research Center
HIT Shenzhen Graduate School
Shenzhen, China
[2]School of Management
Shenyang University of Technology
Shenyang, China
{ linjing, wangxuan, zhangjia } @cs.hitsz.edu.cn

***Abstract*—There exist problems of slow convergence and local optimum in standard Q-learning algorithm. Truncated TD estimate returns efficiency and simulated annealing algorithm increase the chance of exploration. To accelerate the algorithm convergence speed and to avoid results in local optimum, this paper combines Q-learning algorithm, truncated TD estimation and simulated annealing algorithm. We apply improved Q-learning algorithm using into the imperfect information game (SiGuo military chess game), and realize a self-learning of imperfect information game system. Experimental outcomes show that this system can dynamically adjust each weight which describes game state according to the results. Further, it speeds up the process of learning, effectively simulates human intelligence and makes reasonable step, and significantly improves system performance.**

***Keywords*—Q-learning, truncated TD, simulated annealing, imperfect information game**

I. INTRODUCTION

Reinforcement learning (RL) is a promising approach for building autonomous agents that improve their performance from experience. The two most important and distinguishing features of RL are trial-and-error and delayed reward. It includes Temporal Difference proposed by Sutton [1] and Q-learning of Watkins [2]. Those methods are extensively used in many applications, such as game and robot.

Recently, Q-learning algorithm is one of the fast developed RLs. Many tasks can be asymptotically learned by adopting Markov Decision Process (MDP) framework and by using RL techniques. But, in practice, regarding its standard algorithms, there arise two main questions. First, its speed of converge is slow. Second, its exploration strategies often fall into local optimization. Therefore, it is well worth to improve Q-learning in order to find the efficient ways to speed up their convergence and prevent strategies from falling in local optimization. To tackle these problems, we combine Q-learning with TD method to accelerate convergence speed and thus can quickly learn in the initial stages of learning. For second problem, an improved Q-learning algorithm based on Simulated Annealing is proposed in exploration strategies, to avoid the local optimization and blind exploration. In addition, the redundant learning after finding optimal path is avoided, and the time of learning is reduced.

Computing an accurate cost-to-go function is significant and the key to Game. Tradition value function e.g. static linear evaluation can't adjust its action dynamically according to learned acknowledge and it is too slow to promote its performance. Further more, we adopt the improved Q-learning algorithm in SiGuo military chess game [3] [4], which can adjust each weight of state in dynamical based on the result of game. It speeds up the process of learning and modifies performance.

This paper is organized as follows. Q-learning is briefly introduced in Section 2. Section 3 gives a primary introduction of TD method and its improvement, truncated TD method discussed by Watkins and proposed by Cichosz [5]. Specific description about the algorithm with adaptive parameter λ in truncated TD and its application in Q-learning is also presented in Section 3. Section 4 describes exploration strategies in Q-learning by using Simulated Annealing. Finally, Section 5 and Section 6 contains our experiment and some concluding remarks.

II. Q-LEARNING

Q-learning is a format of RL that doesn't need environment model. It can be regarded as a method of asynchronous Dynamic Programming (DP). The basic idea of Q-learning is to update the Q-function at each time period. This function maps each pair of state and action to the expected reward in order to take this action at that state and follows an optimal strategy for all the other future states. In this way, the learned Q-function directly approximates the optimal action-value function [6] without the need to learn a model of the environment explicitly.

Algorithm 1 *Q-learning algorithm*
· *Initialization: Initialize Q(s,a), for every s, a.*
· *Loop: until s_t is a terminal state.*
1. Observe a current state s_t and choose an action a_t, according to greedy strategy then executes it;
2. Observe a new state s_{t+1} and Receive an immediate reinforcement r_t;
3. Update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

$(0 \le \gamma < 1)$;
$s_t \leftarrow s_{t+1}$.

Figure 1. Q-learning algorithm

In the original Q-learning algorithm, the greedy strategy with pure exploitation is used. But by this strategy, it is generally difficult to obtain satisfactory results as algorithm usually falls into a local optimal solution and it is so slow that practical learning tasks are hard to solve successfully. Both of problems will be solved in later section 2 and 3.

In order to avoid huge space of Q-learning look-up table and its extremely slow convergence, the proposed strategy is that the Q-function is implemented by neural network (Fig. 2). Thus, *Q(s,a)* can be described as function of weight *w*. ($w=(w_{ij},v_{jk})$) （$1\le i\le 130, 1\le j\le 5, k=1$）. Then the purpose of Q-learning is to adjust weight *w*.

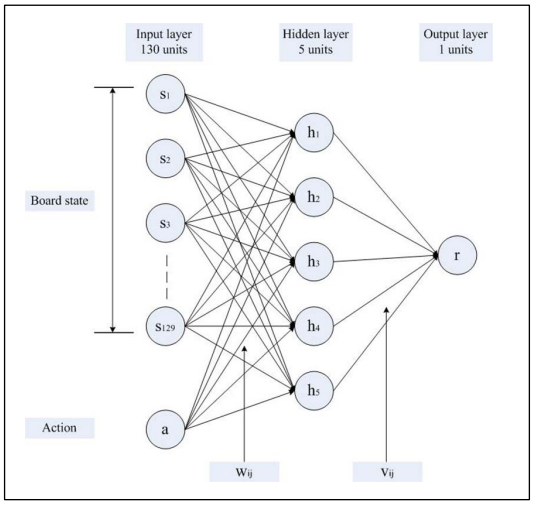

Figure 2. Q-function implemented by neural network

III. Q-LEARNING AND TEMPORAL DIFFERENCE

Standard Q-learning is implemented by look-up table and it updates a Q-function once a time. Not having a decided teaching signal, and only adjusting its action by outside evaluation, results in an endless learning process.

In order to resolve its slow convergence, there are some aspects which should be tackled. Learning strategy and action search strategy is advanced by: accretions of the number of modify Q-function each time; Adoption of multi-step; Combination of TD algorithm. For instance, Peng and Williams propose Q(λ) [7] ,which is a model-free algorithm that extends Q-learning algorithm by combining with TD(λ) returns for general λ in a natural way for delayed reinforcement learning. Their motivation was to estimate TD returns better by the use of TD errors and promote its convergence. TD method provides a way of using the scalar rewards e.g. existing supervised training techniques can be used to tune the function approximator. TD(λ) uses eligibility traces to compute TD returns, which costs large computation, however, truncated TD(λ) [5] shows its computational efficiency advantages in comparison with TD(λ).

In this section, we combine Q-learning and truncated TD(λ). Further more, we employ an adaptive parameter λ. The parameter λ is used to distribute credit throughout sequences of actions, leading to faster learning and also to alleviate the non-Markovian effect of coarse state-space quantization.

A. Truncated TD(λ) return

In Q(λ) algorithm, TD(λ) return (Sutton, 1988) at stage t is

$$z_t^{\lambda} = \sum_{k=0}^{\infty} (\gamma\lambda)^k [r_{t+k} + \gamma(1-\lambda) V_{t+k}(x_{t+k+1})] . \quad (1)$$

For TD(1) return corresponds to $\lambda=1$ and is defined as $z_t^{\lambda} = \sum_{k=0}^{\infty} (\gamma)^k r_{t+k}$. For $\lambda=0$, TD return is the sum of the immediate reinforcement and the discounted estimate value of the successor state. And intermediate λ values provide a smooth heuristic interpolation between these two limits. Unfortunately, the TD(λ) return z_t^{λ} , an infinite series, is not available in practical computation.

The Truncated TD(λ) applies the following formula as the update error at each stage t instead of the traditional TD(λ) error Δ_t^{λ} to the on-line learning of V^*.

$$z_t^{\lambda} - V_t(x_t) = \Delta_t^{\lambda} + \sum_{k=0}^{\infty} (\gamma\lambda)^k [V_{t+k}(x_{t+k}) - V_{t+k-1}(x_{t+k})] \quad (2)$$

Then the m-step truncated TD(λ) return (Cichosz, 1995) defined as

$$z_t^{\lambda,m} = \sum_{k=0}^{m-1} (\gamma\lambda)^k [r_{t+k} + \gamma(1-\lambda) V_{t+k}(x_{t+k+1})] + (\gamma\lambda)^m V_{t+m-1}(x_{t+m}) . \quad (3)$$

Equation (3) is used to approximate TD(λ) return. Correspondingly, the update operation for the new implementation of Q-learning is written as

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha[z_t^{\lambda,m} - Q_t(x_t, a_t)] . \quad (4)$$

B. Adaptive λ in TTD(λ,m) and choice of m

According to the analysis by Cichosz, too large λ reduces the reliability of learning: some runs converged to suboptimal policies. In contrast, λ is so small that it will cause low learning speed and slow convergence. Our experiments in section 5 provide a good illustration of the role of λ in TD learning, which consists in trading off between bias and variance. Besides, λ must be subject to the value of learning rate α. The problem of large λ with large α is that it impairs the reliability of learning and leads to premature convergence to suboptimal resolutions. Further more, the value of learning rate α must guarantee that algorithms can converge correctly. Consequently, it is a reasonable strategy to use adaptive λ and a not large α. For instance, take a larger value of λ in the initial stages of learning so as to quickly collect some useful data and reduce λ gradually to a smaller value as learning proceeds to guarantee an optimal policy. In this work, learning rate α is small and adaptive λ value selection id defined as

$$\lambda \leftarrow \lambda_{low} + \frac{1}{100\alpha N_t}(\lambda - \lambda_{low}) . \quad (5)$$

In (5), λ_{low} anticipates about not so slowly convergence and N_t is a function of time t (i. e. $N_t=a*t+b$).

Another important problem is the choice of m. The value of m, in the same way, depends on λ. For $\lambda=0$, the value of m equals to one. For $\lambda=1$ and $\gamma=1$, m must be large enough to accurately approximate TD(λ), but in this case, $\lambda=1$ is usually not the optimal. Intermediary, we would have a such value of m that the discount $(\gamma\lambda)^m$ is a small number.

C. Combine adaptive λ in TTD(λ,m) with Q-learning

In this subsection, we combine adaptive λ in TTD(λ,m) with Q-learning. Truncated TD(λ) returns are used for on-line learning by keeping track of the last m number of visited states, and updating at each step the predicted utility of the least recent state of those m states. An experience buffer $<s_{[k]}, a_{[k]}, r_{[k]}, q_{[k]}>$ was maintained, where $k=0,1,...,m-1$. For example, $<s_{[0]}, a_{[0]}, r_{[0]}, q_{[0]}>$ means experience buffer in current step t and $< s_{[i]}, a_{[i]}, r_{[i]}, q_{[i]}>$ means experience buffer in step t-i.

Algorithm 2 *TTD -based Q-learning with adaptive λ algorithm*

- *Initialization: Initialize Q(s,a), for every s, a. Initialize λ in random between* $(\lambda_{low}, 1]$.
- *Loop:*
  1. *Observe a current state* s_t, *choose an action* a_t *according to greedy strategy, and then execute it;*
  2. *Observe a new state* s_{t+1} *and receive an immediate reinforcement* r_t.

Save experience buffer $< s_{[0]}= s_t , a_{[0]} =a_t , r_{[0]}= r_t , q_{[0]} = \max_a Q(s_{t+1}, a) >$;

3. *For k=0, 1, ... m-1 do*

   if k=0 then $z = r_{[k]} + \gamma q_{[k]}$

   else $z = r_{[k]} + \gamma(\lambda z + (1-\lambda) q_{[k]})$

4. *Update*

$$Q(s_{[m-1]}, a_{[m-1]}) \leftarrow Q(s_{[m-1]}, a_{[m-1]}) + \alpha[z - Q(s_{[m-1]}, a_{[m-1]})] \qquad s_t \leftarrow s_{t+1}$$

$$\lambda \leftarrow \lambda_{low} + \frac{1}{100\alpha N_t}(\lambda - \lambda_{low})$$

$$< s_{[k+1]}, a_{[k+1]}, r_{[k+1]}, q_{[k+1]} > \leftarrow < s_{[k]}, a_{[k]}, r_{[k]}, q_{[k]} > \qquad (k=0, 1 ...m-2)$$

- *Terminate:* s_t *is a terminal state.*

1. *Observe a current state* s_t, *choose an action* a_t *according to greedy strategy, and then execute it;*
2. *Observe a new state* s_{t+1} *and receive an immediate reinforcement* r_t.

Save experience buffer $< s_{[0]}= s_t , a_{[0]} =a_t , r_{[0]}= r_t , q_{[0]}= 0>$;

3. *For* $k_0=0, 1, ... m-1$ *do*

   a. *For* $k= k_0, k_0+1, ...m-1$ *do*

      if $k= k_0$ *then* $z = r_{[k]} + \gamma q_{[k]}$

      else $z = r_{[k]} + \gamma(\lambda z + (1 - \lambda) q_{[k]})$

   b. *Update*

$$Q(s_{[m-1]}, a_{[m-1]}) \leftarrow Q(s_{[m-1]}, a_{[m-1]}) + \alpha[z - Q(s_{[m-1]}, a_{[m-1]})]$$

$$< s_{[k+1]}, a_{[k+1]}, r_{[k+1]}, q_{[k+1]} > \leftarrow < s_{[k]}, a_{[k]}, r_{[k]}, q_{[k]} > \qquad (k= k_0, k_0+1, ...m-2)$$

Figure 3. TTD-based Q-learning with adaptive λ algorithm

As described in algorithm 2, TTD(λ,m) -based Q-learning is similar to Q-learning when $\lambda=0$ and m tends to infinite.

In the beginning of m steps, experience buffer $<s_{[k]}, a_{[k]}, r_{[k]}, q_{[k]}>$ are constructed and not learning happened. After m steps, large λ and experience buffer promote learning. Gradually, λ is reduced which guarantee the convergence of the algorithm. Compared with TD(λ)-based algorithm, experience buffer $<s_{[k]}, a_{[k]}, r_{[k]}, q_{[k]}>$ can also be used to on-line learning.

IV. Exploration Strategies

The balance between exploration and exploitation is another problem in Q-learning. If there is not a balance between exploration and exploitation, the agent can not learn successfully. The most commonly used balance method is trying to take action, find out the biggest reward, and carry it out. But this method causes the local optimum. There are several common action exploration strategies, such as ε-greedy Strategy and Boltzmann distribution. In this section, Metropolis Criterion of simulated annealing (SA) algorithm is introduced to Q-learning exploration strategy, in order to solve the problem of balance between the exploration and exploitation.

A. Simulated Annealing Algorithm

The SA algorithm is an important branch of the computational processes resembling nature [8] and its motivation is an effective approximate algorithm to resolve combinatorial optimization problems. It is a simulation of the annealing process of solid.

One of the primary techniques, Metropolis Criterion, is evolved from the important sampling method proposed by Metropolis in 1953 [9]. Then Kirkpatrick used the similarity of solid annealing process and combinatorial optimization problem, which was introduced to optimize process to solve the optimal solution and local minima [10]. The SA algorithm generates the sequence of solutions of combinatorial and optimal problem by Metropolis algorithms. For minimization problem, the probability of transition $P_i(i \Rightarrow j)$, corresponds to Metropolis Criterion, determines whether to accept the transfer from the current solution i to the new solution j. The SA algorithm does not reject the worst solution which leaves the local optimal solution. The probability $P_i(i \Rightarrow j)$ is defined as follow:

$$P_i(i \Rightarrow j) = \begin{cases} 1 & if\ f(j) \le f(i) \\ \exp(\dfrac{f(i) - f(j)}{t}) & else \end{cases} \quad (6)$$

B. Q-learning algorithm based on Metropolis Criterion

The main difference between Metropolis-based Q-learning and standard Q-learning is that an agent chooses an action without depending entirely on what have been learned by the optimal strategy. Again, the strategy and randomly selected actions are examined according to the temperature to increase the chance of exploration [11] [12].

Algorithm 3 *Metropolis-based Q-learning algorithm*

- *Initialization: Initialize Q(s,a), for every s, a.*
- *Loop: until* s_t *is a terminal state.*

1. *Observe a current state* s_t *and choose an action* a_q *randomly, set* $a_t \leftarrow a_p$ *according to greedy strategy;*
2. *Generate a random number* $\xi \in [0,1]$,

   if $\xi < exp[Q(s_t, a_q) - Q(s_t, a_p) / Temperature]$ *then* $a_t \leftarrow a_q$

3. *Execute an action* a_t , *observe a new state* s_{t+1} *and receive an immediate reinforcement* r_t;
4. *Update*

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (0 \le \gamma < 1)$$

$s_t \leftarrow s_{t+1}$.

Figure 4. Metropolis-based Q-learning algorithm

In order to make agents to jump out the strategy in local optimum, without using the greedy criterion completely in document [1], ε-greedy strategy is introduced to explore current non-optimal movement in a fixed probability ε. Comparing learned strategies with different ε, $\varepsilon>0$ often gets better results than $\varepsilon=0$. As the learning continues, it is apparently unnecessary to explore in the fixed probability. Further more, it reduces the system performance and even has adverse effect of learning. However, after introducing Metropolis criterion, as the temperature getting lower, the probability of exploration is decreased and almost nonexistent in the final. The improved algorithm is described in Fig. 4 and the experiments as well as performance are introduced in next section.

V. EXPERIMENTS AND PERFORMANCE WITH SIGUO MILITARY CHESS GAME

In this section, the Military Chess game system called SiGuoJunQi is presented, which is based on Q-learning algorithm. First, a brief introduction about the SiGuo Military Chess game is provided.

A. Introduction to siguo Military Chess game

The Military Chess [3][13], which is also called Kriegspiel, has been a very popular game in China for many years, especially with the development of Internet. According to an investigation conducted by Tencent Company [4], which is one of the most famous Internet game operators in China, the number of Military Chess online players exceeds 300,000 at the same time.

The rules of the Military Chess are quite complex. Briefly speaking, the basic process of the game is to move pieces, attack opponents' pieces and finally occupy the position of enemy's flag. Although it sounds quite like Chess, there are at least three main differences between them as following.

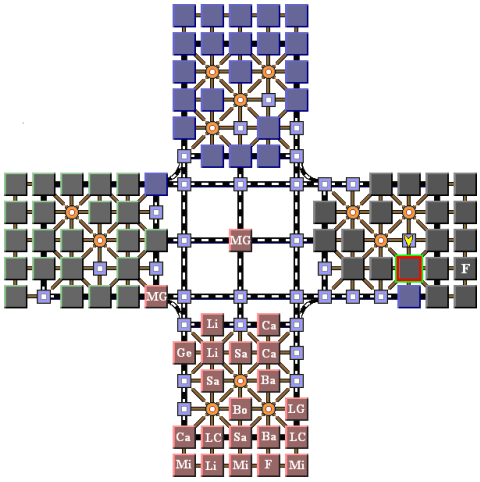

Figure 5. The Interface of SiGuoJunQi

First, each player has an associate who sits in the opposite and two opponents sit aside, like Bridge, so that collaboration should be considered in the game. Second, the pieces which have a certain military rank can only destroy those with lower ranks. Take the piece which has the rank of Major General for example, it has the power to destroy most of the other pieces when they conflicts except the pieces of General and Lieutenant General. The General, which is the highest rank piece on the board, can only be destroyed by Main, or perish together with Bomb or another General. Last but not least important, player can not see the ranks of other players' pieces, even though it belongs to his associate. As a result, players have to judge the ranks of other pieces by the information appearing in the process of game, and sometimes can only make a guess. This is the most difficult and complex point of the development of Military Chess game system.

A Military Chess game system is developed which is called SiGuoJunQi. The game interface is shown in Fig. 5 and Tab. 1 shows the ranks of pieces on the board. The system can operate a game both on single computer and on the Internet.

TABLE I. RANKS OF PIECES ON THE BOARD

Rank of Piece	Abbreviation on Board
General	Ge
Lieutenant General	L．G
Major General	M．G
Brigadier General	B．G
Colonel	Co
Lieutenant Colonel	L．C
Captain	Ca
Lieutenant	Li
Sapper	Sa
Bomb	Bo
Landmine	Mi
Flag	F

B. Experiments and Performance

Improved Q-learning algorithm is used to train the weights of an evaluation function for our SiGuo military chess game. After each end of game, update and adjust parameter weight w in neural network, as follow:

$$w \leftarrow w+\alpha\sum_{t=1}^{N-1}\frac{\partial Q(s_t,a_t)}{\partial w}[\sum_{j=t}^{N-1}\lambda_t^{j-t}z_t] \tag{7}$$

Performance of improved Q-learning algorithm is evaluated at next three aspects. In our experiments, learning rate α equals 0.4 which is based on experience and $\lambda_{best}=0.3$ and $\lambda_{min}=0.1$. Besides, $m=20$ in TTD(λ,m).

In the first group of experiments, we compare the speed of convergence in each Q-learning. As Fig. 6 shows, TTD-based Q-learning precedes Q(λ) in terms of convergence speed, while it has not superior to Q-learning. It can be realized that TTD-based Q-learning with adaptive λ is the best. This situation can be explained that the parameter λ in Q-learning equals to zero. The zero value of λ is interpreted as TD return is the sum of the immediate reinforcement and the discounted estimate value of the successor state. But in TTD-based Q-learning, $\lambda=0.3$ and TD return is defined as $z_t^{0.3}=\sum_{k=0}^{\infty}(0.3*\gamma)^k[r_{t+k}+0.7\gamma V_{t+k}(x_{t+k+1})]$. Regarding TTD-based Q-learning with adaptive λ, the parameter λ promises the balance of information collection and converge.

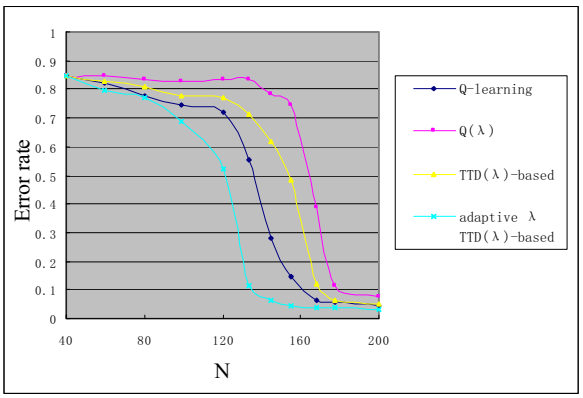

Figure 6. The weight w of neutral network converged along with times of game (N stands for times of game)

Fig. 7 depicts the number of forward nodes from current state used to calculate TD return and vary with the chessboard state. When $\lambda=0$, TD return is the sum of the immediate reinforcement and the discounted estimate value of the successor state and consequently the number of forward node equals to one. In addition, $\lambda=0.3$ matches along with 37 each board state. Especially, the number of forward node corresponds to adaptive λ which is variable. At the beginning of chessboard state, λ is so large that it obtains a significant learning speedup. As (5), parameter λ is reduced step by step. When it is closer to the end, it accelerates convergence.

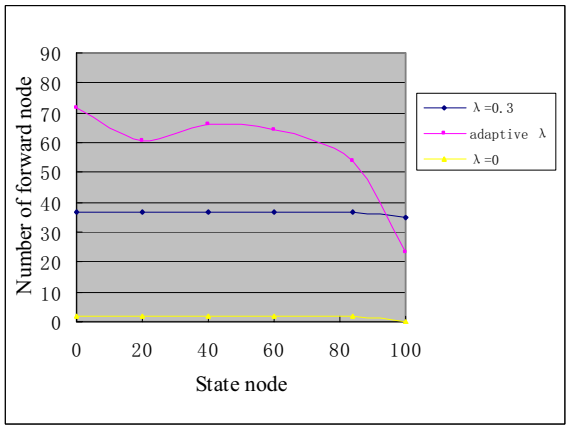

Figure 7. The number of forward node from current state used to calculate TD return changed with the chessboard state (experiment in 100 games)

TABLE II. EXPERIMENT RESULT ABOUT EXPLORATION STRATEGY IN 100 GAMES OF SIGUOJUNQI BASED ON EACH ALGORITHMS

Algorithms	Average Iterations	Number of exploration in non-optimal solution	Win rate
Q-learning	75	0	14%
ε-greedy based Q-learning ($\varepsilon=0.1$)	87	9	17%
Metropolis-based Q-learning	79	6	19%

The Second experiment is operating in the case of exploration strategy. In Tab. 2, we can see that, in terms of the total number of exploration in non-optimal solution, Metropolis-based Q-learning is less in contrast with the high rate of winning. As we know, both ε-greedy based Q-learning and Metropolis-based Q-learning are not greedy algorithms. Moreover, both the algorithms increase the probability of exploration in non-optimal solution. Metropolis-based Q-learning achieves relatively good winning rate as Compare to ε-greedy based Q-learning. It can be explained, as the knowledge accumulates, agent explores in the fixed parameter $\varepsilon=0.1$, which reduces the system performance and even has adverse effect of learning in ε-greedy based Q-learning. Whereas in Metropolis-based Q-learning, agent explores at the probability almost equal to one and decreases probability as learning continue.

Finally, the most concerning experiment is about the winning rate of the improved Q-learning by combining the TTD(λ)-based Q-learning and Metropolis-based Q-learning. From Tab. 3, we can conclude that the new improved Q-learning applied to SiGuo military chess game has achieved a progress.

TABLE III. EXPERIMENT RESULT ABOUT WIN RATE IN 100 GAMES OF SIGUOJUNQI BASED ON EACH ALGORITHMS

Algorithms	Win	Draw	lose	Win rate
Q-learning	15	78	7	15%
TTD(λ)-based Q-learning ($\alpha=0.3$,λ is adaptive)	19	80	1	19%
Metropolis Criterion based Q-learning	20	77	3	20%
Metropolis Criterion and TTD(λ)-based Q-learning ($\alpha=0.3$,λ is adaptive)	23	72	5	23%

VI. CONCLUSION

In this paper, we have presented improved Q-learning by combining the standard Q-learning, TTD(λ) and simulated annealing. Thereafter, the applications of the improved Q-learning into SiGuo military chess game with the implementation of a self-learning imperfect information game system have been illustrated. Finally the experiment, in which a military chess evaluation functions was trained by on-line game against a mixture of human and computer opponents, showed a satisfactory result.

Theoretical aspects has already shown that the improved Q-learning converge faster than simple Q-learning. Further more, it avoids suboptimum and repeated learning. It can also be applied to other intelligent control systems.

ACKNOWLEDGMENT

This work is supported by the National High-tech R&D Program of China (863 Program, No. 2007AA01Z194).

REFERENCES

[1] Sutton, R S. "Learning to predit by the method of temporal differences," Machine Learning, 1988, pp.9-44

[2] Watkins C J C H, Dayan P. "Q-learning," Machine Learning, 1992, pp.279-292

[3] Jiajia Zhang, Xuan Wang, Jin Lin, "UCT Algorithm in Imperfect Information Multi-Player Military Chess Game," JCIS 2008

[4] http://www.tencent.com/

[5] Cichosz P. “Truncating temporal differences: On the efficient implementation of reinforcement learning,” Journal of Artificial Intelligence Research,1995, pp.287-318

[6] D. P. Bertsekas and J. N. Tsitsiklis. “Neuro-Dynamic Programming,” Athena Scientific, 1995,pp.298

[7] Peng, J. & Williams, R. J. “Incremental multi-step Q-learning,” In Proceedings of the Eleventh International Conference on Machine Learning. Morgan Kaufmann, 1994

[8] S. Boettcher and A. Percus, “Nature’s way of optimizing,” Artific. Intell., 2000, pp.275-286

[9] N.Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E.Teller, “Equation of calculations by fast computing machines,” J. Chem.Phys.,1953, pp.1087–1092

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” Science, 1983, pp.671-680

[11] Guo Rui, Peng Jun,Wu Min, “The Application of Reinforcement Learning in Nondeterministic MDPs Policy Finding Question,” Computer Engineering and Applications, 2005, pp.36-38

[12] Chen Sheng-lei, “Metropolis Policy-based Multi-step Q Learning Algorithm and Performance Simulation,” Journal of System Simulation, 2007, pp.1284-1287

[13] XIA Z. Y. , HU Y. A. ,“Analyze and guess type of piece in the computer game intelligent system,” Lecture Notes in Computer Science, 2005, pp.1174-1183