Theoretical and Applied Informatics
ISSN 1896–5334
Vol.24 (2012), no. 2
pp. 159–171
DOI: 10.2478/v10179-012-0008-1

Prioritized epoch-incremental Q-learning algorithm

ROMAN ZAJDEL

Department of Electrical Engineering and Informatics
Rzeszow University of Technology
al. Powstańców Warszawy 12, Rzeszów, Poland
e-mail: rzajdel@prz-rzeszow.pl

Received 19 April 2012, Revised 19 June 2012, Accepted 22 June 2012.

Abstract: The basic reinforcement learning algorithms, such as Q-learning or Sarsa, are characterized by short time-consuming single learning step, however the number of epochs necessary to achieve the optimal policy is not acceptable. There are many methods that reduce the number of‘ necessary epochs, like TD(λ>0), Dyna or prioritized sweeping, but their computational time is considerable. This paper proposes a combination of Q-learning algorithm performed in the incremental mode with the method of acceleration executed in the epoch mode. This acceleration is based on the distance to the terminal state. This approach ensures the maintenance of short time of a single learning step and high efficiency comparable with Dyna or prioritized sweeping. Proposed algorithm is compared with Q(λ)-learning, Dyna-Q and prioritized sweeping in the experiments of three grid worlds. The time-consuming learning process and number of epochs necessary to reach the terminal state is used to evaluate the efficiency of compared algorithms.

Keywords: reinforcement learning, Q-learning, grid world

1. Introduction

The efficiency of the basic reinforcement learning algorithms, e.g. Q-learning [18], AHC [2] or Sarsa [11], measured in number of epochs to obtain the optimal policy, is relatively small. For this reason, there is a small number of practical implementations of these algorithms to more complex problems. The unquestionable advantage of these algorithms is a small computational complexity which implies the extremely short learning time of each epoch. The short learning time is essential for applications of the reinforcement learning algorithms to the on-line control problems. Thus, the acceleration methods of reinforcement learning algorithms should ensure both relatively small computational complexity and high efficiency. Unfortunately, the acceleration methods used up to date, that reduce the number of epochs needed to obtain the optimal policy, require

significantly more learning time. For example, one of the most often used acceleration method, like temporary-differences mechanism TD(λ>0), requires the additional memory elements known as eligibility traces. The learning time of TD(λ>0) significantly grows, because the updates of the policy function are made for all states, not only for one (actual) state, like in the class of basic TD(0) reinforcement learning algorithms [15, 18].

The learning methods, such as Dyna [13, 14] or prioritized sweeping [8, 9] which are much more efficient than TD(λ>0), also belong to the class of memory based methods. The basic idea behind these methods is the use of the adaptive environment model in reinforcement learning. Their efficiency is much better than in the case of TD(λ>0), but it is at the expense of the considerable increase of learning time than in the case of TD(λ>0). The main idea of these algorithms is to compute a fixed number of updates of V value function or Q action-value function for respectively: the states or the state-action pairs which were active in the past. Furthermore, in the case of the Dyna algorithm, these update states are chosen randomly. In the case of the prioritized sweeping, the update states are prioritized by the temporary differences update error.

In this paper, the epoch-incremental reinforcement learning algorithm is proposed in order to obtain the highly efficient learning method with very small learning time. The main idea of this algorithm is to combine the simplest form of Q-learning algorithm (1-step Q-learning) that is performed in incremental mode with the acceleration method performed in epoch mode. Proposed acceleration method is based, to a large extent, on the reinforcement signal which is obtained when the terminal state is reached. Next, this signal is back-propagated to all visited states, as signaled by their non-zero state transition probability. The proposed algorithm is compared to three well known and often used reinforcement learning algorithms: Q(λ)-learning, Dyna-Q and prioritized sweeping. These algorithms are used to solve control problem of several grid worlds.

The other version of epoch-incremental reinforcement learning algorithm was proposed in [20]. That solution was different from the one presented in this paper. Firstly, the environment model was based on the model regarded in Dyna-learning and prioritized sweeping algorithms. In this article, the environment model utilizes the transition probability. Secondly, the acceleration of the learning process in the epoch mode used the reinforcement signal stored in the model. In this contribution, the terminal reinforcement signal is back-propagated to all visited states.

The organization of the paper is as follows. Section 2 contains the brief overview of the basic reinforcement learning algorithms such as Q(0)-learning, Q(λ)-learning, Dyna-Q and prioritized sweeping. Section 3 describes the proposed prioritized epoch-incremental Q-learning algorithm. Section 4 compares the proposed method with the algorithms highlighted in Section 2 in the problem of the popular Sutton grid world [13]. Section 5 provides the details on the computational experiment setup in two grid worlds and discuses the results. Section 6 concludes the paper with some final remarks.

2. Reinforcement learning

Reinforcement learning addresses the problem of an agent that must learn to perform a task through trial and error interaction with an unknown environment [15]. The agent and the environment interact continuously until the terminal state is reached. The agent senses the environment throughout its sensors and, based on its current sensory inputs, selects an action to perform in the environment. Depending on the effect of its action the agent obtains a reward. The agents goal is to maximize the discounted sum of future reinforcements r_t received in the long temporary horizon, what is usually formalized as $\sum_{t=0}^{\infty} \gamma^t r_t$, where $\gamma \in [0, 1]$ is the agent's discount rate.

2.1. Q-learning

There exist different types of reinforcement learning algorithms. The Q-learning proposed by Watkins [18] is probably the best known. This algorithm computes, by successive approximations, a table of all values $Q(s, a)$, called the Q-table which represents the expected payoff that agent can obtain in state s after it performs action a. The Q-table is updated according to the following formula:

$$Q(s,a) = Q(s,a) + \beta(r + \gamma \max_{a'} Q(s',a') - Q(s,a)), \tag{1}$$

where the maximization operator refers to the action value a' which may be performed in next state s', and $0 < \beta \leq 1$ is the learning rate.

The basic Q-learning algorithm can be significantly improved considering the history of states' activations represented by the eligibility traces. The eligibility trace is parameterized by recency factor $\lambda \in [0, 1]$, therefore this enriched learning method is called Q(λ)-learning. The Watkins's proposition [18, 15] of combining the eligibility traces and the Q-learning leads to the following update method:

$$Q(s,a) = Q(s,a) + \beta(r + \gamma \max_{a'} Q(s',a') - Q(s,a))e(s,a), \tag{2}$$

where the eligibility trace for the state-action pair (s, a) is determined as:

$$e(s,a) = \gamma\lambda e(s,a) + \delta(s,a), \tag{3}$$

in which:

$$\delta(s,a) = \begin{cases} 1 & \text{if } s \text{ is a visited state and } a \text{ is an executed action,} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

The eligibility trace of each state becomes large after state activation and then it decreases exponentially until the state is visited again [3].

2.2. Dyna-Q and prioritized sweeping

Dyna and prioritized sweeping require a model of the environment in order to improve the policy represented by Q-table. Given a state and an action, a model produces a prediction of a next (resultant) state and a reward.

Dyna-Q is the Q-learning based algorithm which simultaneously uses the experience to build the model and to adjust the policy. Additionally, it uses the model to adjust the policy. Dyna-Q operates in a loop of interactions with the environment as follows. After each transition $(s, a) \rightarrow (s', r)$, the model is stored in its table entry for the argument (s, a). On the basis of (s,a) it is possible to predict (s', r). Then, the Q value at the state s and the action a is updated using rule (1). Afterwards, some number (denoted further as N) of additional updates is performed on the basis of the model. Each of these updates consists of the random choice of the state action pair (s, a) that has been experienced before, and the query to the model with this pair. The model returns the next state s' and the reward r as its prediction. Finally, the update according to (1) is performed. A reasonable value of N can be determined based on the relative speeds of computation and of taking action [5].

The prioritized sweeping is similar to the Dyna-Q algorithm, except that the updates are not chosen at random but depending on the priority p which is the absolute value of temporal differences error $|r + \gamma \max_{a'} Q(s', a') - Q(s, a)|$. If priority p is greater than the arbitrary defined threshold θ then the pair (s, a) is stored into the queue (called *PQueue*) with the priority. Next, only N states with the highest priority are updated according to (1) using the model, similarly to Dyna-Q.

3. Prioritized epoch-incremental Q-learning

The fundamentals of the proposed algorithm are based on the idea of Q(λ)-learning, Dyna-Q and prioritized sweeping. These fundamentals utilize the following four observations.

If the prioritized sweeping algorithm is used in the grid world with the absorbing state, during the first learning episode, only the state-action pair leading directly into the goal state has the priority p greater than zero. That means that it is unnecessary to do all the updates, except the last, that leads to the goal state. This suggests, that the updates could be performed backwards from the goal states which produce the strongest reinforcement signal.

The second observation refers to the time when the update process of Q-table is performed. Namely, if the N updates are performed incrementally, the learning time significantly increases, which is extremely essential in use of this algorithm in control problems, like the cart-pole problem [2], the ball-beam system [19] or the mobile robot. Thus, the adaptive control algorithm, based on reinforcement method, should ensure

short time needed to complete learning process. An obvious solution, consisting in the reduction of the number of N updates, may contribute to the decrease of the learning efficiency.

Thirdly, the Dyna and the prioritized sweeping algorithms are based on the adaptive model of the environment which, at the beginning of the learning process, is unreliable. That is because it is built on the basis of too few observation. The credibility of the model increases as the next episodes are performed, and it is maximal when the last learning episode is finished. Therefore, it is strongly advisable to utilize this final model.

The last observation refers to the real-world application of the reinforcement learning algorithms. Namely, after ending of each learning episode, the system is not restored to the starting state instantly, but in nonzero time. This time can be used to perform updates which, in Dyna and prioritized sweeping algorithms, are performed in the incremental mode, i.e. before the learning episodes are finished.

1. Initialize $Q(s,a)$, $N(s,a)$ and $N(s,a,s')$ arbitrarily for all $s \in S$ and $a \in A$
2. Repeat (for each episode):
3. Initialize s
4. Repeat (for each step of episode):
5. Choose action a in s using policy derived from Q (e.g., ε -greedy)
6. Execute action a, observe reward r and resultant state s'
7. $Q(s,a) \leftarrow Q(s,a) + \beta(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
8. $N(s,a) \leftarrow N(s,a) + 1;\ N(s,a,s') \leftarrow N(s,a,s') + 1$
9. $p(s,a,s') \leftarrow N(s,a,s') / N(s,a)$
10. $s \leftarrow s'$
11. until s is TERMINAL
12. $PQueue \leftarrow (s = TERMINAL, a = 0, d = 0)$
13. while $PQueue_d$ is not empty
14. $d \leftarrow d + 1$
15. for each $s' \in PQueue_{d-1}$
16. for each $a \in A$, for each $s \in preds(s') \cap (not\ in\ PQueue)$
17. if $p(s,a,s') > 0$, $PQueue \leftarrow (s,a,s',d)$
18. Repeat for all (except first) elements of PQueue
19. $s,a,s',d \leftarrow PQueue$
20. $Q(s,a) \leftarrow Q(s,a) + \alpha(r(\gamma\lambda)^d + \gamma \max_{a'} Q(s',a') - Q(s,a))$

Fig. 1. Prioritized epoch-incremental Q-learning algorithm with priority determined by estimated value of the distance to the terminal state

The complete proposed algorithm presented in form of the pseudocode is given in Fig. 1. All four observations mentioned above are applied here. At the beginning, the Q-table and the tables needed by the model are initialized arbitrary. The learning process consists of some number of episodes. Each of them starts from initial state, that is often the same in each episode. Then, the loop begins, in which the basic Q-learning algorithm is first performed (steps 5-7). The information about the possible transition to state s' after executing action a in state s is stored in the model table $p(s, a, s')$, where $N(s, a)$ denotes the number of time units in which action a is executed in s, and $N(s, a, s')$ is the number of time units resulting in the transition to state s' [1, 8, 9, 16] (steps 8 and 9). The incremental part of learning algorithm is performed until the terminal state (the goal state in grid world) is reached. Then, starting from the terminal state, the queue (*PQueue*) of every visited state-action pair is maintained and prioritized by the minimum number of time steps d to the terminal state (steps 12 and 13). The *preds*(s') is the set of all states which have been observed as immediate predecessors of state s'. Thereafter, for all 4-tuples (s, a, s', d) put in the *PQueue*, one performs the following update of the action-value function (step 20):

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r(\gamma\lambda)^d + \gamma \max_{a'} Q(s', a') - Q(s, a)). \tag{5}$$

In the above formula, *r* is the reward signal associated with the achievement of the terminal state which, in exponentially decreasing manner $(\gamma\lambda)^d$, is propagated backwards to the states which contributed to its achievement.

The inspiration of the multiplication of the reinforcement signal by $\gamma\lambda$ coefficient is the observation of the Q(λ)-learning algorithm performance, in which the eligibility trace of state-action pairs was decreased by this factor in each iteration (3). It is worth noticing that in Q(λ)-learning method, all the elements of the action-value function are actualized in the degree which depends on the distance from the actual state. In case of the proposed prioritized epoch-incremental Q-learning algorithm, the actualizations are performed only for these elements of action-value function Q which are responsible for suboptimal strategy determined on the basis of the shorted distance to the absorbing state.

Summing up, the proposed algorithm is combination of 1-step tabular Q-learning, that is performed in incremental mode and the form of learning from "goal state" executed in epoch mode.

4. Case Study – A Navigation Task

This section presents the effect of operation of epoch-incremental Q-learning algorithm in the grid world. Grid worlds are often used for comparing how quickly different reinforcement learning methods converge towards to a stable solution [4, 5, 7, 8, 9, 10,

12, 13, 14, 17]. The navigation task is the maze shown in the Fig. 2. The maze is a 6 by 9 grid of states which includes the starting state and the goal state marked 'S' and 'G', respectively. The dark states are obstacles and cannot be entered. Agent can move in one of four possible directions: LEFT, RIGHT, UP, and DOWN.

0	0	0	0	0	0	0		G
0	0		0	0	0	0		0
S	0		0	0	0	0		0
0	0		0	0	0	0	0	0
0	0	0	0	0		0	0	0
0	0	0	0	0	0	0	0	0

Fig. 2. The Sutton's 6x9 grid world with 46 reachable states [13]

The reward is zero for all transitions except for these into the goal state 'G', for which it is 1. If the goal state is reached, the learning episode ends and the agent is transported to the start state to begin the next trial [13].

1 → 14	2 → 13	3 → 12	4 ↓ 11	5 ↓ 10	6 ↓ 9	7 ↓ 8	8	9 **G** 0
10 ↓ 15	11 ↑ 14	12	13 → 10	14 ↓ 9	15 → 8	16 ↓ 7	17	18 ↑ 1
19 ↓ **S** 14	20 ↓ 13	21	22 → 9	23 ↓ 8	24 ↓ 7	25 ↓ 6	26	27 ↑ 2
28 ↓ 13	29 ↓ 12	30	31	32 → 7	33 → 6	34 → 5	35 → 4	36 ↑ 3
37 → 12	38 → 11	39 → 10	40 → 9	41 ↑ 8	42	43 ↑ 6	44 ↑ 5	45 ↑ 4
46 ↑ 13	47 ↑ 12	48 ↑ 11	49 ↑ 10	50	51 → 8	52 ↑ 7	53 ↑ 6	54 ↑ 5

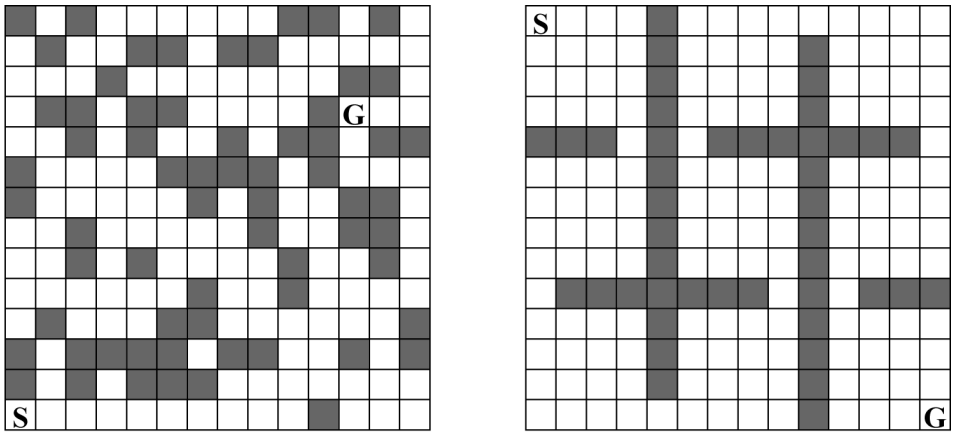
Fig. 3. The policy and the distance to the terminal state 'G' represented by the queue *PQueue* after first episode

The exemplary effect of queued procedure (step 13, Fig. 1) performed after first episode is shown in Fig. 3. In the top-left corner of each cell the number of the state is placed. The distance d is located in the bottom-right corner. The arrows represent the actions which allow to reach the absorbing state after the smallest number of steps (step 16, Fig. 1). One can also notice, that the *PQueue* visualized in this way is part of the optimal policy. Therefore, the updates of Q-table based on *PQueue* (step 18, Fig. 1) should ensure improvement of the policy. Lack of an arrow in the box 31 and 50 implies that these states are not visited during this episode.

Similar, distance based algorithm is presented by Peng and Williams [9], but it has two significant differences. Firstly, Peng and Williams suggest the measure of the distance to the start state, not to the terminal state, as it is proposed in this paper. The main objective of the work presented in [9] is to try to assess the effect of any updates on the estimated long-term reward at the start state. Secondly, in [9], the modification of the action-value function on the basis of the distance is performed in the incremental mode. In this work, this modification is performed in epoch mode. The proposed algorithm, on the basis of the terminal state, estimates the shortest distance d between all past active states and the terminal state. Moreover, the semioptimal policy is determined with the use of the probability $p(s,a,s')$. On the basis of this semioptimal policy and the distance d, the action-value function is modified.

5. Empirical results

Three different grid worlds are used in this work: 6x9 (Fig. 2) [13], 14×14 (Fig. 4. (a)) [8] and 14×14 (Fig. 4. (b)) [20]. The minimal number of agent's moves from 'S' to 'G' is 14, 25 and 70 respectively.

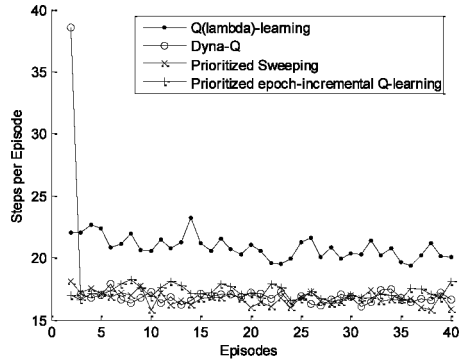


(a) (b)

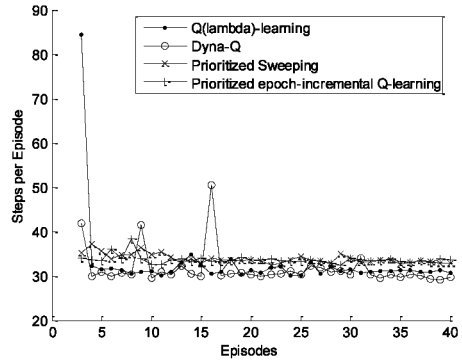
Fig. 4. The 14×14 grid worlds with 130 (a) and 161 (b) reachable states

The proposed prioritized epoch-incremental Q-learning algorithm is compared with three well-known methods: Q(λ)-learning, Dyna-Q, and prioritized sweeping. The initial values of $Q(s,a)$, $N(s,a)$ and $N(s,a,s')$ are zeros, the step size is β =*0.1*, and the exploration parameter ε =*0.1*.

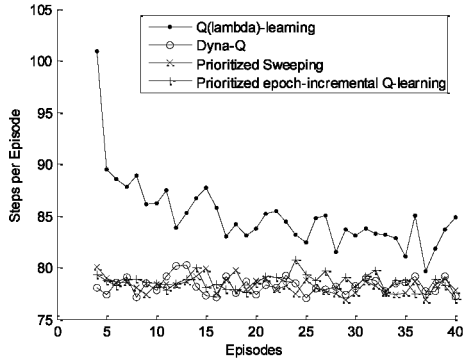
The number of the updates for Dyna-Q and prioritized sweeping in 6x9 grid world [2] and in 14×14 grid world [20] is set to N =50 and N =100, respectively. The



(a)



(b)



(c)

Fig. 5. The average learning curves for the grid world with 46 (a), 130 (b) and 161 (c) reachable states, respectively. One trip from the start state ‘S’ to the goal state ‘G’ denotes the episode

learning curves show the number of steps taken by the agent in each episode averaged over 30 repetitions of the experiments (Fig. 5). The first step episode in 6×9 grid world, first two steps per episode in Moore and Atkeson 14×14 grid world and first three steps per episode in the 14×14 grid world with 161 states are significantly bigger than the remaining ones, therefore they are omitted on the Fig. 5.

After a few first episodes, the number of steps is stabilized and, except for the Q(λ)-learning shown in Fig. 5(c), does not decrease. The average number of steps over all plotted points is shown in Tab. 1.

Algorithm Environment	**Q(λ)-learning**	**Dyna-Q**	**prioritized sweeping**	**prioritized epoch-incremental Q-learning**
46 states min 14	20.861	17.379	16.722	17.225
130 states min 25	32.744	31.624	33.865	33.664
161 states min 70	85.069	78.271	78.283	78.600

Tab 1. The average number of steps necessary to reach the terminal state 'G' from start state 'S'. In the left column, under the number of grid worlds reachable states, the minimal number of moves from 'S' to 'G' is shown

Because the ε-greedy action selection method is used (ε =0.1), the minimal numbers of achieved moves from 'S' to 'G' are about 10% greater than the perfect result, shown in left column. The Q(λ)-learning algorithm appears to be the worst for 46 and 161 states grid worlds. The 130 states grid world is different from the others, because from the start state to the goal state there are only two possible paths. The agent does not have too much choice as in the other grid worlds and if the optimal policy is found, it is remembered. Therefore, for this grid world the results for all four algorithms are similar. For the remaining grid worlds (46 and 161 states), the average number of steps per episode for proposed prioritized epoch-incremental Q-learning is comparable with the Dyna-Q and prioritized sweeping.

Algorithm Environment	**Q(λ)-learning**	**Dyna-Q**	**prioritized sweeping**	**Prioritized epoch-incremental Q-learning**	
				incremental and epoch	Incremental
46 states	4.1%	14.3%	100.0%	9.0%	2.0%
130 states	1.4%	4.5%	100.0%	11.9%	0.8%
161 states	2.7%	6.8%	100.0%	7.5%	0.5%

Tab 2. The average time runs calculated as a percentage of the maximal time

The average time of a single learning step is shown in Tab. 2. The time is calculated as percentage of the maximal learning time in order to become independent from performance of computer system. The prioritized sweeping is characterized by

the longest learning time. It is mainly the consequence of the prioritizing the state-action pairs according to the time differences error. The execution time of the prioritized epoch-incremental Q-learning algorithm is shown in two columns. Column 5, signed as 'incremental and epoch', contains the average time of all steps of the learning algorithm (step 2 in Fig. 1). This time is comparable with the Dyna-Q computational time. From the practical point of view, the more significant issue is the incremental mode time (step 2.2 in Fig. 1) which is shown in column 6 of Tab. 2. One can notice that this time is shorter not only for the Dyna-Q but also for the Q(λ)-learning algorithm in all three grid worlds.

6. Conclusions

In this paper, the new method of acceleration of reinforcement learning algorithm is proposed. The main idea of this method is to extract the part of algorithm which is responsible for acceleration of learning and to execute it after ending the incremental learning stage, that is in epoch mode. In this way both the incremental and the total (incremental and epoch) learning time is meaningfully shortened (see Tab. 2). Furthermore, the efficiency of proposed algorithm, measured in number of epochs to obtain the optimal path from start state to the goal state is comparable with the efficiency of Dyna-Q and prioritized sweeping algorithms (see Tab. 2 and Fig. 5). The decrease of the execution time of the incremental mode algorithm is relevant in practical applications of the reinforcement learning algorithms.

Author will adapt the proposed algorithm to continuous-state environment in order to apply it to a mobile robot control, a cart pole system and a ball-beam system. The Takagi-Sugeno system, fuzzy CMAC and radial basis function neural network will also be used to approximate the action-value function.

Acknowledgments

This work was supported by Polish Ministry of Science and Higher Education under the grant 3745/B/T02/2009/36.

References

1. A. Barto, S. Bradtke, S. Singh: *Learning to Act using Real-Time Dynamic Programming*, Artificial Intelligence, Special Volume on Computational Research on Interaction and Agency, 72 (1), pp. 81-138, 1995.
2. A. Barto, R. Sutton, C. Anderson: *Neuronlike adaptive elements that can solve difficult learning problem*, IEEE Trans. SMC, 13, pp. 834-847, 1983.

3. P. Cichosz: *Systemy uczące się*, WNT, Warszawa, 2000 (in Polish).
4. P. Crook, G. Hayes: *Learning in a State of Confusion: Perceptual Aliasing in Grid World Navigation*, Proc. of Towards Intelligent Mobile Robots, 2003.
5. L. Kaelbing, M. Litman, A. Moore: *Reinforcement Learning*: *A Survey*, Journal of Artificial Intelligence Research, 4, pp. 237-285, 1996.
6. P. Lanzi: *Adaptive Agents with Reinforcement Learning and Internal Memory*, Proc. of the Sixth International Conference on the Simulation of Adaptive Behavior (SAB'2000), pages 333-342. The MIT Press, Cambridge, MA, 2000.
7. J. Loch, S. Singh: *Using eligibility traces to find the best memoryless policy in partially observable markov decision processes*, In ICML, pp. 323-331, 1998.
8. A. Moore, C. Atkeson: *Prioritized sweeping: Reinforcement learning with less data and less time*, Machine Learning, 13, pp. 103-130, 1993.
9. J. Peng, R. Williams: *Efficient learning and planning within the Dyna framework*, Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior, Hawai, pp. 281-290, 1993.
10. M. Pickett, A. Barto: *PolicyBlocks: An Algorithm for Creating Useful Macro-Actions in Reinforcement Learning*, Proc. of the International Conference on Machine Learning, 19. pp. 506-513, 2002.
11. G. Rummery, M. Niranjan: *On line q-learning using connectionist systems*, Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.
12. A. Sherstov, P. Stone: *Improving Action Selection in MDP's via Knowledge Transfer*, Proc of the Nation Conference on Artificial Intelligence, 20(2), pp. 1024-1029, 2005.
13. R. Sutton: *Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming*, Proc. of Seventh Int. Conf. on Machine Learning, pp. 216-224, 1990.
14. R. Sutton: *Planning by incremental dynamic programming*, Proc. of the Ninth Conference on Machine Learning, pp. 353-357, 1991.
15. R. Sutton, A. Barto: *Reinforcement learning: An Introduction*, MIT Press, Cambridge, 1998.
16. P. Tadepalli, D. Ok: *Model-Based Average Reward Reinforcement Learning*, Artificial Intelligence, 100, pp. 177-224, 1998.
17. B. Tanner, S. Sutton: *Temporal-Difference Networks with History*, Proc. of the 2005 International Joint Conference on Artificial Intelligence, pp 865-870, 2005.
18. C. Watkins: *Learning from delayed Rewards*, PhD thesis. Cambridge University, Cambridge, England. 1989.
19. P.E. Wellstead: *Introduction to Physical System Modelling*, Control System Principles, 2000.
20. R. Zajdel: *Epoch-Incremental Queue-Dyna Algorithm*, Lecture Notes in Artificial Intelligence 5097, pp. 1160-1170, 2008.

Priorytetowany epokowo-inkrementacyjny algorytm Q-learning

Streszczenie

Efektywność podstawowych algorytmów uczenia ze wzmocnieniem Q-learning i Sarsa, mierzona liczbą prób niezbędnych do uzyskania strategii optymalnej jest stosunkowo niewielka. Stąd też możliwości praktycznego zastosowania tego algorytmu są niewielkie. Zaletą tych podstawowych algorytmów jest jednak niewielka złożoność obliczeniowa, sprawiająca, że czas wykonania pojedynczego kroku uczenia jest na tyle mały, że znakomicie sprawdzają się one w systemach sterowania online. Stosowane metody przyśpieszania procesu uczenia ze wzmocnieniem, które pozwalają na uzyskanie stanu absorbującego po znacznie mniejszej liczbie prób, niż algorytmy podstawowe powodują najczęściej zwiększenie złożoności obliczeniowej i wydłużenie czasu wykonania pojedynczego kroku uczenia. Najczęściej stosowane przyśpieszanie metodą różnic czasowych TD(λ>0) wiąże się z zastosowaniem dodatkowych elementów pamięciowych, jakimi są ślady aktywności (*eligibility traces*). Czas wykonania pojedynczego kroku uczenia w takim algorytmie znacznie się wydłuża, gdyż w odróżnieniu od algorytmu podstawowego, gdzie aktualizacji podlegała wyłącznie funkcja wartości akcji tylko dla stanu aktywnego, tutaj aktualizację przeprowadza się dla wszystkich stanów.

Bardziej wydajne metody przyśpieszania, takie jak Dyna, czy też prioritized sweeping również należą do klasy algorytmów pamięciowych, a ich główną ideą jest uczenie ze wzmocnieniem w oparciu o adaptacyjny model środowiska. Metody te pozwalają na uzyskanie stanu absorbującego w znacznie mniejszej liczbie prób, jednakże, na skutek zwiększonej złożoności obliczeniowej, czas wykonania pojedynczego kroku uczenia jest już istotnym czynnikiem ograniczającym zastosowanie tych metod w systemach o znacznej liczbie stanów. Istotą tych algorytmów jest dokonywanie ustalonej liczby aktualizacji funkcji wartości akcji stanów aktywnych w przeszłości, przy czym w przypadku algorytmu Dyna są to stany losowo wybrane, natomiast w przypadku prioritized sweeping stany uszeregowane wg wielkości błędu aktualizacji.

W niniejszym artykule zaproponowano epokowo-inkrementacyjny algorytm uczenia ze wzmocnieniem, którego główną ideą jest połączenie podstawowego, inkrementacyjnego algorytmu uczenia ze wzmocnieniem Q-lerning z algorytmem przyśpieszania wykonywanym epokowo. Zaproponowana metoda uczenia epokowego w głównej mierze opiera się na rzeczywistej wartości sygnału wzmocnienia obserwowanego przy przejściu do stanu absorbującego, który jest następnie wykładniczo propagowany wstecz w zależności od estymowanej odległości od stanu absorbującego. Dzięki takiemu podejściu uzyskano niewielki czas uczenia pojedynczego kroku w trybie inkrementacyjnym (Tab. 2) przy zachowaniu efektywności typowej dla algorytmów Dyna, czy też prioritized sweeping (Tab. 1 i Fig. 5).