Informatics in Education, 2007, Vol. 6, No. 2, 255–268

Investigation of Q-Learning in the Context of a Virtual Learning Environment

Dalia BAZIUKAITĖ
Department of Computer Science, Klaipėda University
Herkaus Manto 84, 92294 Klaipėda, Lithuania
e-mail: dalia.baziukaite@ik.ku.lt

Received: July 2006

Abstract. We investigate the possibility to apply a known machine learning algorithm of Q-learning in the domain of a Virtual Learning Environment (VLE). It is important in this problem domain to have algorithms that learn their optimal values in a rather short time expressed in terms of the iteration number. The problem domain is a VLE in which an agent plays a role of the teacher. With time it moves to different states and makes decisions which regarding action to choose for moving from current state to the next state. Some actions taken are more efficient than others. The transition process through the set of states ends in a final (goal) state, one which provides the agent with the largest benefit possible. The best course of action is to reach the goal state with the maximum return available. This paper introduces a way of definition of a rewards matrix, which allows the maximum tolerance for the changes of a discounted reward value to be achieved. It also proposes way of an application of the Q-learning that allows a teaching policy to exist, which maps the situation in the learning environment.

Key words: learning algorithms, reinforcement, convergence, virtual environment.

1. Introduction

A virtual learning environment (VLE) is defined as the web-based application suitable for information exchange between teacher and learner (Hobbs, 2002), as well as between learners themselves. There exist another subset of VLEs that differs from the previous in the ability of sequencing the curriculum. Those VLEs are known as adaptive (Kinshuk, 2002; Weber, 1997; Iglesias, 2003; Metcalfe, 2001; Baziukaitė, 2006). The adaptivity is implemented in the systems based on the approach used in the ELM-ART II (Weber, 1997) adaptive tutoring system that supports learning programming in LISP. Another subset of VLEs is intelligent environmets. Intelligent environments use methods of Artificial Intelligence (AI) to solve the problem of student modeling and curriculum sequencing (Iglesias, 2003). In an intelligent VLE the role of teacher is prescribed to the agent, who makes decisions how to teach the learner depending on the experience presented to the agent.

Reinforcement learning is the process during which the agent improves its behavior in the environment (Sutton, 1998; Kaelbling, 1996; Gosavi, 2004; Littman, 1996) using

the experiences it has received from interacting with an environment. The experiences have the form of tuple $\langle x, a, y, r\rangle$, with state x, action a, resulting state y and scalar immediate reward r. The Markov Decision Process (MDP) model is a way of formalizing the reinforcement learning problem. A finite MDP is defined by the tuple $\langle S, A, P, R\rangle$, with a finite set of states S, a finite set of actions A, a transition function P, and a reward function R. The optimal behavior for an agent in an MDP depends on an optimallity criterion. The optimality criterion could be expressed as infinit-horizon expected discounted total-reward (Sutton, 1998; Kaelbling, 1996) or infinit-horizon expected average reward (Gosavi, 2004). For both cases the optimal behavior can be found by identifying the optimal value function defined recursively by

$$V^*(x) = \max_a \Big(R(x,a) + \gamma \sum_y P(x,a,y)V^*(y)\Big), \tag{1}$$

in the case of infinit-horizon expected discounted total-reward, where $0 \leqslant \gamma < 1$ is a discount parameter that controls the degree to which future rewards are significant compared to immediate rewards, or by

$$V^*(x) = \max_a \Big(R(x,a) - \rho^* + \sum_y P(x,a,y)V^*(y)\Big), \tag{2}$$

in the case of infinit-horizon expected average reward, where ρ^* is an average reward.

Reinforcement Learning (RL) is a type of Machine Learning (ML), and is also a branch of AI. It allows machines and software agents to automatically determine ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn its behavior. This feedback is called a *reinforcement signal*. There are many different algorithms that address this issue. RL is defined by a specific type of problem, and all its solutions are classified as Reinforcement Learning algorithms. In the problem, an agent is supposed to decide the best action to select based on its current state. When this step is repeated, the problem is known as a *Markov Decision Process*.

RL allows the machine or software agent to learn its behavior based on feedback from the environment. This behavior can be learned once and for all, or keep adapting as time goes on. If the problem is modeled with care, some RL algorithms can converge to the global optimum. This is the ideal behavior that maximizes the reward. This automated learning scheme implies that there is little need for a human expert who knows about the domain of application. As mentioned, there are many different solutions to the problem. The most popular allow the software agent to select an action that will maximize the reward in the long term, and not only in the immediate future. Such algorithms are know to have *infinite horizon*. In practice, this is done by learning to estimate the value of a particular state. This estimate is adjusted over time by propagating part of the next state's reward. If all the states and all the actions are tried a sufficient amount of times, this will allow an optimal policy to be defined. The action that maximizes the value of the next state is selected (Gosavi, 2004; Littman, 1996).

The next section shortly describes value iteration and policy iteration algorithms that are used to solve the MDP problem. Section 3 will explain the Q-learning algorithm, which is being investigated in the following sections. Section 4 presents a general scenario depicting how the algorithm can serve in finding a policy for teaching a given learner. Section 5 brings up results of the investigation and explains a discovered way of convenient definition of a rewards matrix. Finally, we will conclude by summarizing the achieved results and giving possible guidance to future work.

2. Markov Decision Processes

Eqs. 1, 2 are called Bellman's equations for discounted reward and average reward case correspondingly. In the rest of the paper we will concentrate on the case of the infinit-horizon expected discounted reward. Bellman's equation formalizes the overall goal of the Markov Decision Process. The goal is to find a function, called a policy, which specifies which action to take in each state, so as to maximize some function of the sequence of rewards (Gosavi, 2004; Littman, 1996). Markov Decision Process is defined by a set of possible states of the agent $X = \{x_1, x_2, \ldots, x_n\}$, a set of actions $A = \{a_1, a_2, \ldots, a_m\}$, a set of rewards $R = \{r_1, r_2, \ldots, r_n\}$ and a transition probability function $P_{ij}^k = Pr\{x_{t+1} = y | x_t, a_t\}$, where $i, j = 1..n$, $i \neq j$, and $k = 1..m$. Here the MDP serves as a way to solve the Reinforcement-learning problem. Reinforcement-learning is the problem of getting an agent to act in the world (environment) so as to maximize its rewards. The environment is modeled as a stochastic finite state machine with inputs and outputs (Murphy, 2002). Bellman's equation for defined MDP can be solved using value iteration or policy iteration algorithms.

We rewrite the Eq. 1 in the form of recursive iterations as follows

$$V^{n+1}(x_i) = \max_a \Big(r_i + \gamma \sum_{j=1}^{n} P_{ij}^k V^n(x_j) \Big). \tag{3}$$

The value iteration algorithm solves the given MDP by computing $V^n(x_i)$ for all i until converged. The algorithm has converged when

$$\max_i \left| V^{n+1}(x_i) - V^n(x_i) \right| < \epsilon.$$

The optimal policy is then defined as

$$\arg\max_a \Big(R(x,a) + \gamma \sum_y P(x,a,y) V^*(y) \Big). \tag{4}$$

Using policy iteration algorithm the $\pi(x_i)$ defines an action selected in the i'th state and is called a policy. Then the algorithm solves the given MDP computing $V^n(x_i)$ for all i using $\pi(x_i)$ and finding

$$\pi_k(x_i) = \arg\max_a \Big(r_i + \gamma \sum_j P_{ij}^k V^n(x_j)\Big). \tag{5}$$

The algorithm keeps computing until $\pi_k = \pi_{k+1}$. If this condition holds the optimal policy has been found. The algorithms of value iteration and policy iteration require an explicit aproximation of R and P. Unlike those two algorithms, the Q-learning algorithm allows an optimal policy to be found without explicitly aproximating R and P (Kaelbling, 1996; Gosavi, 2004). The Q-learning algorithm estimates the optimal Q function

$$Q^*(x,a) = R(x,a) + \gamma \sum_y P(x,a,y)V^*(y). \tag{6}$$

Knowing the $Q^*(x,a)$ the optimal value function is found from $V^*(x) = \max_a Q^*(x,a)$. Given the experience at step t $\langle x_t, a_t, y_t, r_t\rangle$ and the current estimate $Q_t(x,a)$ the Q-learning updates to the next step

$$Q_{t+1}(x_t,a_t) := \big(1-\alpha_t(x_t,a_t)\big)Q_t(x_t,a_t)+\alpha_t(x_t,a_t)\Big(r_t+\gamma \max_a Q_t(y_t,a)\Big). \tag{7}$$

Littman and Szepesvari (Littman, 1996) have shown that the Q-learning algorithm converges to the optimal Q function with probability 1 over $X \times A$.

3. Q-Learning Algorithm

Q-learning is value-iteration based on RL. An RL model consists of an environment, a learning agent, a set of actions and a response from the environment. The knowledge base of an agent is made up from *Q-factors* for each state-action pair. We can view numerical values of Q-factors in terms of costs or rewards. In the first case the algorithm tends to minimize Q-factors for each state-action pair, and in the second case it tends to maximize them. Our case of application will be based on maximization of Q-factors.

Before learning begins, the values $Q(x,a)$ for all states x and all actions a are set to the same value. When the system is in a decision-making state x, the learning agent examines the Q-factors for all actions in state x and selects the action $x^{'}$ with the minimum Q-factor (if values are in terms of cost). This leads the system along a path untill the system encounters another decision-making state (y). While traveling over this path, i.e. a state-transition from x to y, which is simulated in the simulator, the agent gathers information from the environment about the immediate costs incurred and the time spent during the statechange. This information is used by the agent with the help of its learning algorithm to update the factor $Q(x,y)$. A poor action results in an increase of this value while a good action that results in low cost causes the value to be decreased (or increased

if the value is in terms of rewards). Of course the exact change is determined by the algorithm which is developed from the Bellman equation (Eq. 1, Eq. 2). In other words, the performance of an action in a state serves as an experience for the agent which is used to update it knowledge base. Thus every piece of experience makes the agent a trifle smarter in its perception of the environment than it was before. As all state-action pairs are encountered (theoretically an infinite number of times), the agent learns the optimal actions in each state (Gosavi, 2004; Szepesvari, 1997; Jaakkola, 1994; Littman, 1996).

Now we come to a precise formulation of an algorithm. We will provide a version of the learning algorithm for the infinit-horizon expected discounted total reward (Gosavi, 2004). This actually is a value-iteration for Eq. 1.

Algorithm. *Q-learning for infinit-horizon expected **discounted** total reward.*

For all states x and actions a
  set $Q(x,a) = 0$.
Let iteration count $k = 0$.
Set discount factor γ to some value $(0 \leqslant \gamma < 1)$.

While $k < ITERMAX$[a] do
  1. Compute α_k[b] $= \frac{N_{StateVisit}}{k}$[c] .
  2. With probability $\frac{1}{|\mathcal{A}(x)|}$ select action a in state x that maximizes the Q-factor $Q(x,a)$; otherwise choose a random (exploratory) action from the set $\{\mathcal{A}(x) \backslash a\}$.
  3. Simulate the chosen action. Let the next state be denoted by y and $r(x,a,y)$ denote the transition reward.
  4. Update $Q(x,a)$ by $Q(x,a) \leftarrow (1-\alpha_k)Q(x,a) + \alpha_k(r(x,a,y) + \gamma Q_{max}(y,a))$.
  5. Set current state x to new state y; and increase k by $k \leftarrow k+1$.

For each state x declare the action a, for which Q-factor is maximum
  to be the optimal action.

Output: optimal policy.

[a] ITERMAX should be a large integer.

[b] α_k is a learning rate.

[c] $N_{StateVisit}$ is the number of times the given state x is visited.

The Q-learning algorithm can be viewed as a stochastic process to which techniques of stochastic approximation are generally applicable. The proof of the convergence of Q-learning is available in (Jaakkola, 1994).

4. General Scenario for an Application

To solve the issue of training an agent to apply the optimal policy in order to teach a student with the view of the largest benefit we apply the Q-learning algorithm. For that, we need to describe our system in terms of Markov Decision Process (MDP). We introduce four possible states of a Teacher agent.

The first possible state we entitle Beginner; the second, Preintermediate; the third, Intermediate; and the fourth, Advanced, with the meaning that the agent knows the learner has beginner, preintermediate, intermediate, and advanced knowledge levels in the defined Curricullum (Baziukaitė, 2002; Baziukaitė, 2003; Baziukaitė, 2006). Actions that the agent could take in each state are enumerated as follows: the first, give self-test with feedback showing correct answers (A1); the second, give self-test with feedback (A2); the third, give self-test without feedback (A3). The rewards that the agent gets after transition to the corresponding state are primarily defined as 0, 5, 10 and 20. The probabilistic set of possible states of the agent is depicted in Fig. 1. Our goal is to find an optimal policy for the teacher in the terms of reinforcement learning, which will show the actions of the teacher that are best to apply as a teaching method in order to maximize the future rewards and to reach the goal state as quick as possible. In this case the goal state is the fourth possible state of the teacher agent. Solving the given problem we apply the Q-learning algorithm. After iterative computation the matrix of Q-factors is computed.

Fig. 1 shows the situation in transition between possible states. Transitions between these states map the Markov property, which can be referenced in subsection 3.5 of (Sutton, 1998). The probabilities shown on the arrows are only needed if the problem would be solved using value iteration (see Eq. 3) or policy iteration (see Eq. 4) algorithms. In order to solve this problem by applying Q-learning we have to specify rewards, discount

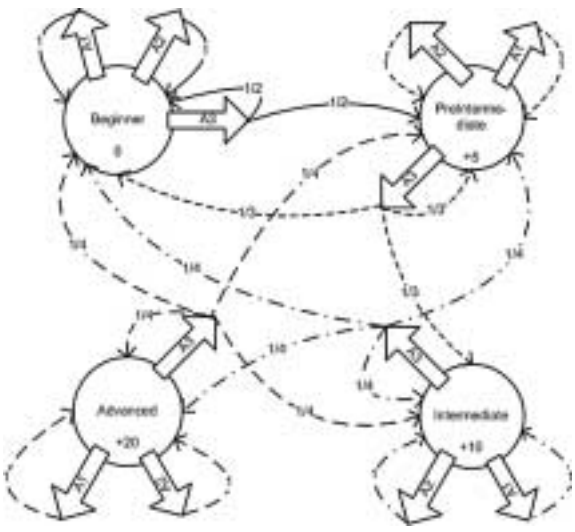

Fig. 1. The probabilistic set of possible states of the Teacher agent.

factor and compute the matrix of Q-factors. This matrix has a number of rows equal to the number of the possible states of the agent and a number of columns equal to the number of actions available. In our case this is an 4×3 matrix. An action to take first in the learning phase of the algorithm is defined stochastically by generating a random value. This means that values of Q-factors in the matrix Q may vary in the different runs of the algorithm. For example, after the routine run of the algorithm we got the following matrix of Q-factors:

$$Q = \begin{pmatrix} 162.9087 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 176.9087 \\ 0.0000 & 186.9087 & 0.0000 \\ 0.0000 & 0.0000 & 186.9087 \end{pmatrix}. \tag{8}$$

There exist a set of parameters that controls the performance of the algorithm. This set is defined by three elements

$$ParSet = \{R, \gamma, \alpha\},$$

where R is a matrix of rewards the agent receives in each state, γ is a discount factor and α is a learning rate. The given matrix of Q-factors was computed when $R = \{0, 5, 10, 20\}$, $\gamma = 0.9$ and $ITERMAX = 30$.

From this matrix we find the optimal value function $V^*(x)$, which defines the optimal policy for our agent. Optimal value function selects the maximum Q-factors in each row of the given matrix. The maximum Q-factor has a meaning that corresponding action (the number of the column) is the best in the corresponding state (the number of the row). According to the given Q-matrix the optimal policy for the agent is defined by the mapping from states to actions $\pi(x)$: $X \to A$ as is shown bellow.

$$\begin{array}{lcl} \pi(x)\text{: } X & \to & A \\ 1\ (Beginner) & & 1(A1) \\ 2\ (PreIntermediate) & & 3(A3) \\ 3\ (Intermediate) & & 2(A2) \\ 4\ (Advanced) & & 3(A3) \end{array} \tag{9}$$

5. Investigation and Results

Some routine procedures were performed in order to scrutinize the impact of given parameters to the convergence of the algorithm and provide some resulting conclusions.

Having in mind the domain of an application it might become important that algorithm reaches the optimal values for each possible state in a small number of iterations independently from a value assigned to the γ, when $0 \leqslant \gamma < 1$.

Next, we present the results achieved by describing two interrelated experiments.

Experiment 1. *How does the definition of matrix R impact the behavior of the algorithm*?

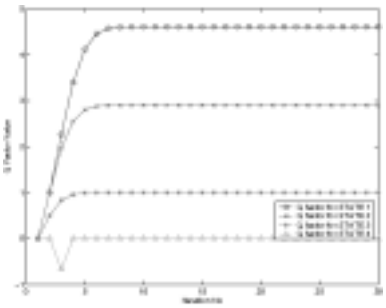

Fig. 2. $\gamma = 0.2$, R as defined in Eq. 10.

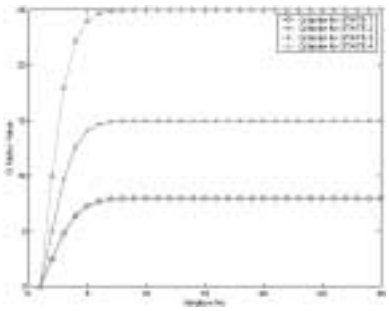

Fig. 3. $\gamma = 0.2$, $R = \{0, 5, 10, 20\}$.

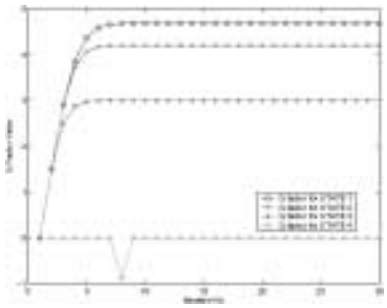

Fig. 4. $\gamma = 0.4$, R as defined in Eq. 10.

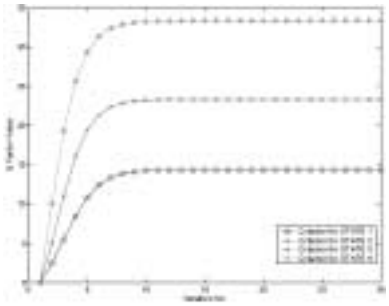

Fig. 5. $\gamma = 0.4$, $R = \{0, 5, 10, 20\}$.

The rewards matrix R can be defined in two ways. It could be defined as a vector, showing the reward value for each possible state, or as $m \times n \times m$ dimension matrix, where m is a number of possible states and n is a number of possible actions.

Experiment 2. *How does the γ impact the convergence of the algorithm*?

During this experiment we investigate the γ parameter value to the algorithm in the given application. We compare results obtained by changing $\gamma = [0.1, \ldots 0.9]$.

Description of the experiments. We will compare results of the algorithm obtained after the matrix R was defined in two ways we have just described. Let us remember the possible actions that could be taken by the agent in each state. By $A1$ we denote action in which agent initiates *self-test with feedback showing correct answers*, by $A2$ – *self-test with feedback* and by $A3$ – *selftest without feedback*. The possible states of being Beginner, Preintermidiate, Intermidiate and Advanced, for more simplicity, we define as $S1$, $S2$, $S3$ and $S4$ accordingly. Then the defined matrix $R_{4\times3\times4}$ will have a meaning as is shown in Tables 1 and 2.

The tables should be read for example, if the learner is in a state Beginner and action $A1$ is taken after which the learner moves to the state Beginner, then give a reward -1. According to the Table 1, the rewards matrix R we define as is shown in Eq. 10.

Such a definition of R allows individual reward to be given depending on what had happened after an action was taken. This way of definition looks more complex in comparison with a method with value vector, but, as the investigation will show, it allows wanted properties to be achieved according to the rate of algorithm convergence.

Table 1
Definition of rewards matrix for current states $S1$ and $S2$

Resulting State	Current State $S1$	Current State $S2$
Beginner ($S1$)	action $A1$ reward -1	action $A1$ reward -2
	action $A2$ reward 0	action $A2$ reward -1
	action $A3$ reward 0	action $A3$ reward 0
Preintermediate ($S2$)	action $A1$ reward 1	action $A1$ reward -1
	action $A2$ reward 2	action $A2$ reward 0
	action $A3$ reward 3	action $A3$ reward 0
Intermediate ($S3$)	action $A1$ reward 2	action $A1$ reward 1
	action $A2$ reward 3	action $A2$ reward 2
	action $A3$ reward 4	action $A3$ reward 3
Advanced ($S4$)	action $A1$ reward 3	action $A1$ reward 2
	action $A2$ reward 4	action $A2$ reward 3
	action $A3$ reward 5	action $A3$ reward 4

Table 2
Definition of rewards matrix for current states $S3$ and $S4$

Resulting State	Current State $S3$	Current State $S4$
Beginner ($S1$)	action $A1$ reward -3	action $A1$ reward -4
	action $A2$ reward -2	action $A2$ reward -3
	action $A3$ reward -1	action $A3$ reward -2
Preintermediate ($S2$)	action $A1$ reward -2	action $A1$ reward -3
	action $A2$ reward -1	action $A2$ reward -2
	action $A3$ reward 0	action $A3$ reward -1
Intermediate ($S3$)	action $A1$ reward -1	action $A1$ reward -2
	action $A2$ reward 0	action $A2$ reward -1
	action $A3$ reward 0	action $A3$ reward 0
Advanced ($S4$)	action $A1$ reward 1	action $A1$ reward -1
	action $A2$ reward 2	action $A2$ reward 0
	action $A3$ reward 3	action $A3$ reward 0

Results of a routine investigation are presented in Figs. 2–9, where x axis is the iteration number and y axis represents the value of the Q-factor. We sign R^* the rewards matrix defined as $R = \{0, 5, 10, 20\}$ and R the rewards matrix defined as is in Eq. 10.

Figs. 2–9 graphically depict the behavior of Q-factors curves for each possible state when different γ values are taken. From these figures we can clearly see that if rewards matrix is defined as R^*, then γ value becomes very important to the convergence rate of the algorithm. For example, in Figs. 3–5 γ has small value ($\gamma \leqslant 0.5$), the Q-factor value reaches its optimal value for each state approximatly in the 12th iteration. In the case when $\gamma > 0.5$ (Figs. 7–9) it reaches those values only in the iteration with a very high

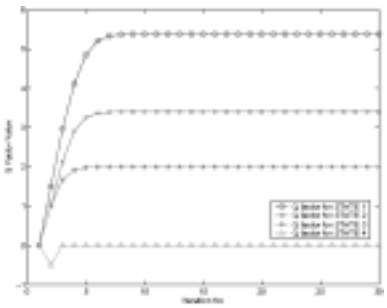

Fig. 6. $\gamma = 0.7$, R as defined in Eq. 10.

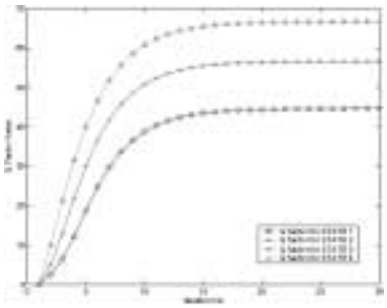

Fig. 7. $\gamma = 0.7$, $R = \{0, 5, 10, 20\}$.

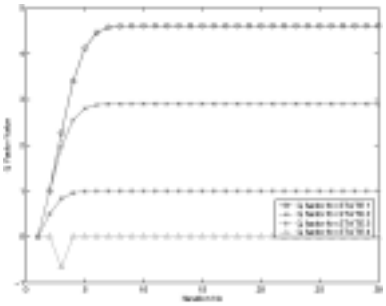

Fig. 8. $\gamma = 0.9$, R as defined in Eq. 10.

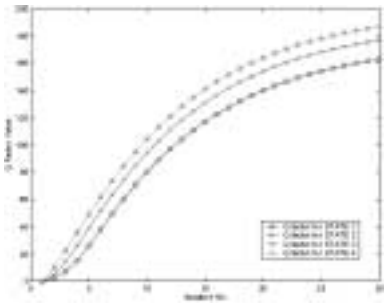

Fig. 9. $\gamma = 0.9$, $R = \{0, 5, 10, 20\}$.

number (~ 70 and higher).

The situation is completely different if rewards matrix is defined as R. In this case the γ value no longer has such a big inpact to the convergence rate of the algorithm. This allows the algorithm to learn optimal Q values in a small number of iterations in a given problem domain, even having high γ values.

$$
\begin{aligned}
R(:,:,1) &= \begin{pmatrix} -1 & 0 & 0 \\ -2 & -1 & 0 \\ -3 & -2 & -1 \\ -4 & -3 & -2 \end{pmatrix} \qquad R(:,:,2) = \begin{pmatrix} 1 & 2 & 3 \\ -1 & 0 & 0 \\ -2 & -1 & 0 \\ -3 & -2 & -1 \end{pmatrix} \\
R(:,:,3) &= \begin{pmatrix} 2 & 3 & 4 \\ 1 & 2 & 3 \\ -1 & 0 & 0 \\ -2 & -1 & 0 \end{pmatrix} \qquad R(:,:,4) = \begin{pmatrix} 3 & 4 & 5 \\ 2 & 3 & 4 \\ 1 & 2 & 3 \\ -1 & 0 & 0 \end{pmatrix}
\end{aligned} \tag{10}
$$

After many routine checks we conclude that:

1. The definition of rewards matrix has a strong impact to the convergence rate of the algorithm in the case of high γ values ($0.5 < \gamma < 1$).
2. High γ values makes convergence of algorithm slower, if R is defined as a vector.
3. The convergence rate of the algorithm remains uniform in the whole interval of γ values, if R is defined as a $m \times n \times m$ dimension matrix.

4. Small γ values $(0 < \gamma \leqslant 0.5)$ do not affect the convergence rate in both cases of definition of R.

PROPOSITION. The best choice for the current application would be to collect information about the quality of the activity. The quality depends on happenings after accomplishing it. Depending on this the reward values in matrix R must change and an agent should be trained in some time intervals or after some amount of rewards changes. After an agent is newly trained the rewards changes count should be reset. In such a way the policy that maps the situation in the learning environment is found.

Algorithm. *Learning the policy*

INPUT: initial $R_{m\times n\times m}$ matrix.
rcount = 0[a]
DO Q-learning with initial$R$ and $\gamma = 0.9$
WHILE true
    IF learner l_i arrives THEN
      BEGIN
        observe state S^*[b]
        apply action a^* and observe state $S^{'}$[c]
      END
    IF learner l_i MOVES to state $S^{'} > S^*$ THEN
      BEGIN
        increase $R(S^*, a^*, S^{'})$
        rcount = rcount + 1
      END
    ELSE
      BEGIN
        decrease $R(S^*, a^*, S^{'})$
        rcount = rcount + 1
      END
    IF rcount == 20 THEN
      BEGIN
        DO Q-learning with $R(:,:,:)$ and $\gamma = 0.9$
        RESET rcount
      END
END
OUTPUT: policy mapping the situation in the learning environment.

[a] $rcount$ is the number of times of rewards changes
[b] S^* is current state
[c] $S^{'}$ is a state a learner moves to after the action a^* was taken

6. Conclusion

The investigations done have shown that the behavior of the tutor could be modeled applying the agent paradigm. For that purpose possible states, actions of an agent and rewards it receives are introduced. The task of training the agent to apply the optimal learning strategy is formulated as the problem of RL and the Q-learning algorithm is applied for finding such a policy. Some previous results are also published in (Baziukaitė, 2004).

The investigation of Q-learning algorithm has shown that the definition of rewards matrix plays a great role in behavior of Q-factors curve that are computed for each possible state. A way of defining a rewards matrix as $m \times n \times m$ dimension matrix, where m is a number of possible states and n is a number of possible actions, is introduced. According to the obtained results that come out after the routine experiments with the algorithm, the following conclusions are formulated:

1. The definition of rewards matrix has a strong impact to the convergence rate of the algorithm in the case of high γ values ($0.5 < \gamma < 1$).
2. High γ values makes convergence of algorithm slower, if R is defined as a vector.
3. The convergence rate of the algorithm remains uniform in the whole interval of γ values, if R is defined as a $m \times n \times m$ dimension matrix.
4. Small γ values $(0 < \gamma \leqslant 0.5)$ do not affect the convergence rate in both cases of definition of R.

As well, the pseudocode describing the flow of actions that allow to train agent to find and apply optimal policy for teaching particular learner in the learning environment is provided. The proposed way of an application of the Q-learning allows a teaching policy to exist, which maps the situation in the learning environment.

References

Baziukaitė, D., A.A. Bielskis, O. Ramašauskas (2002). Applying adaptive learning principles for the e-studies. *Liet. Matem. Rink.*, **42**(spec. issue), 214–218.

Baziukaitė, D. (2003). Concept of adaptive based virtual learning environment. In D. Rutkauskienė (Ed.), *Proceedings of the International Conference TELDA'03*. Kaunas University of Technology, Kaunas, pp. 63–66.

Baziukaitė, D. (2004). Making virtual learning environment more intelligent: application of Markov decision process. *Liet. Matem. Rink.*, **44**(spec. issue), 797–801.

Baziukaitė, D. (2006). Approach to an adaptive and intelligent learning environment. In *CISSE 2005 Proceedings*. Springer (accepted for publication).

Gosavi, A. (2004). Reinforcement learning for long-run average cost. *European Journal of Operational Research*, **155**, 654–674.

Hobbs, D.L. (2002). A constructivist approach to web course design: a review of the literature. *International Journal on E-Learning*, April–June, 60–65.

Iglesias, A., P. Martinez and F. Fernandez (2003). An experience applying reinforcment learning in a web-based adaptive and intelligent educational system. *Informatics in Education*, **2**(2), 223–240.

Jaakkola, T., M.I. Jordan and S.P. Singh (1994). On the convergence of stochastic Iterative dynamic programming algorithms.

Kaelbling, L.P., M.L. Littman and A.W. Moore (1998). Reinforcement learning: a survey. *A Journal of Artificial Intelligence Research*, **4**, 237–285.

Kinshuk, H.H., and A. Patel (2002). Adaptivity through the use of mobile agents in web-based student modelling. *International Journal on E-Learning*, July–September, 55–64.

Levy, A.Y., and D.S. Weld (2000). Intelligent Internet systems. *Artificial Intelligence*, **118**, 1–14.

Littman, M.L., and C. Szepesvari (1996). A generalized reinforcement-learning model: Convergence and applications. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 310–318.

Metcalfe, A., M. Snitzer and J. Austin (2001). Virtual adaptive learning environment. In *IEEE International Conference on Advanced Learning Technologies* (*ICALT'01*), Madison, Wisconsin, 06–08 August, pp. 7-10.

Murphy, K. (2002). Markov decision process toolbox for Matlab.
`http://www.ai.mit.edu/ murphyk/Software/MDP/mdp.html`

Szepesvari, C., and M.L. Littman (1997). Generalized Markov decision processes: Dynamic-programming and Reinforcement-learning algorithms. *Technical Report*, CS-97-05, Brown University.

Sutton, R.S., and A.G. Barto (1998). *Reinforcement Learning*: *An Introduction*. MIT Press, Cambridge.

Weber, G., and M. Specht (1997). User modeling and adaptive navigation support in www-based tutoring systems. In A. Jameson, C. Paris and C. Tasso (Eds.), *User Modeling*: *Proceedings of the Sixth International Conference*, *UM97*. Springer, Wien, New York, pp. 289–300.

D. Baziukaitė recieved her MCs degree in mathematics from Klaipėda University. Currently she is working towards her PhD degree in computer science and mathematics at Klaipėda University. Her research is focused on adaptivity, intelligence, and decision making processes in virtual learning environments. She is a member of the National Association of Distance Education and Lithuanian Society of Mathematicians.

Q-mokymosi algoritmo tyrimas virtualios mokymo(si) aplinkos kontekste

Dalia BAZIUKAITĖ

Straipsnyje aptarta galimybė virtualios mokymo(si) aplinkos (VMA) sąlygomis taikyti Q-mokymosi algoritmą. Šioje taikymų srityje svarbu turėti algoritmus, kurie apsimoko pakankamai greitai, kai apsimokymo laikas yra išreikštas iteracijų skaičiumi. Taikymų sritis yra VMA, kurioje programinis agentas atlieka kuratoriaus vaidmenį. Bėgant laikui agentas pereina skirtingas būsenas priimdamas sprendimą apie tai, kokią veiklą perėjimui į naują būseną reikia parinkti. Vienos veiklos yra vertinamos geriau nei kitos. Perėjimo per būsenas procesas užsibaigia tikslo būsenoje, buvimas kurioje agentui duoda geriausią suminę atlygių grąžą. Straipsnyje taip pat pristatytas atlygių matricos apibrėžties būdas, kuris leidžia suteikti algoritmui aukštą tolerancijos laipsnį parametro reikšmės, nusakančios atlygių nuvertėjimo įtaką ateities būsenoms, pasikeitimui.