Control and Cybernetics

vol. **29** (2000) No. 4

An application of decision rules in reinforcement learning

by

A. Michalski

Institute of Computing Science, Poznań University of Technology,
Piotrowo 3a, 60-965 Poznań
e-mail: michalsk@sol.put.poznan.pl

Abstract: In this paper an application of decision rules to function representation in reinforcement learning is described. Rules are generated incrementally by method based on rough set theory from instances recorded in state-action-Q-value memory. Simulation experiment investigating the performance of the system and results achieved are reported.

Keywords: rough set theory, decision rules, reinforcement learning, Q-learning.

1. Introduction

A reinforcement learning system (also called agent) learns by experimentation and does not require a teacher for proposing correct actions for all possible states the system may find itself in. The agent searches and tries out different actions for every state it encounters and selects the most useful ones according to its current knowledge. This search and selection process is guided by a reinforcement signal that is a performance evaluation feedback. Thus, unlike supervised learning, reinforcement learning is a class of problems where only very simple evaluative information, as opposed to instructive information, is available for learning (Sutton and Barto, 1998). The current knowledge of the agent is represented as a real-valued function defined over state space or state-action pairs space. It is required for large spaces to use function approximator in order to save memory and allow generalization. The necessity of using efficient function representation methods is especially important in problems with continuous space. There are several common methods of approximating learning function described in literature. Best known are: lookup table (Watkins and Dayan, 1992), neural networks (Barto et al., 1983), CMAC (Watkins, 1989), memory-based function approximators (McCallum, 1996; Santamaria et al., 1996). One must note that much of the successful research in reinforcement learning has

lookup tables. Thus, most of them use representations, which require numerical computation to determine the agents' next action. However, it may be necessary to provide agents with symbolic representation to enable them, for instance, to exchange knowledge.

This paper presents another function approximation approach based on decision rules, which establish symbolic representation. Rules are generated according to the method of rough set theory (Skowron, 1995).

The rest of the paper is organized as follows. Section 2 presents a well-known reinforcement learning algorithm called *Q-learning*. In the subsequent section our learning method is described. Section 4 characterizes an evaluation experiment and provides results achieved in comparison with earlier approaches. The last section contains final conclusions and remarks.

2. Standard *Q-learning*

Reinforcement learning is a machine learning paradigm that approximates the conventional optimal control technique known as dynamic programming. Within this paradigm we have controller called the agent and the external world (also called environment) modeled as a discrete-time, finite state, Markov decision process. Each action of an agent is associated with a reward. The task of reinforcement learning is to maximize the long-term discounted reward per action. Our studies are based on the reinforcement learning agent that uses the one-step *Q-learning* algorithm (Watkins, 1989). In this algorithm, the agent decision policy is determined by *state-action* value function, Q, which estimates long-term discounted rewards for each state-action pair. However, the agent does not have such Q-function readily available from the beginning and it must learn such function using its own experience. Thus, the idea is to provide the agent with an initial estimate of the Q-function and let it decide the best action based on the current estimate. Then, the agent can use the outcome of each action (reward received) to asymptotically improve the estimate towards the optimal action value. More specifically, given a current state x and the available actions a_i, a Q-learning agent selects each action a using current, non-optimal estimate of the state-action value function. The action selected is the action leading to the best value at the current state. However, in order to enable the state space exploration the action is choosen randomly according to a given probability distribution. The agent than executes the chosen action, receives an immediate reward r and moves to the next state y. In each time step, the agent updates $Q(x,a)$ (improves estimate for state-action pair (x,a)) by recursively discounting future utilities (next state-action values) and weigthing them by a positive learning rate α:

$$Q(x,a) \leftarrow Q(x,a) + \alpha(r + \gamma V(y) - Q(x,a))^1$$

[1] $f(x_1,\ldots,x_n) \leftarrow f(x_1,\ldots,x_n) + \delta$ means that the value of function f for arguments

Here γ $(0 \leq \gamma < 1)$ is a discount factor, and $V(y)$ is given by:

$$V(y) = \max_{b \in \{a_i\}} Q(y, b).$$

State-action value $Q(x, a)$ is updated only when taking action a from state x but random action selection ensures that each action will be evaluated repeatedly.

As the agent explores the state space, its estimate Q improves gradually, and, eventually, each state value, $V(x)$, approaches: $E\{\sum_{n=1}^{\infty} \gamma^{n-1} r_{t+n}\}$, the discounted expected sum of rewards. Here, r_t is the reward received at time t due to the action chosen at time $t-1$. It was shown in Watkins and Dayan (1992) that Q-learning algorithm converges to an optimal decision policy for a finite Markov decision process and Q-function represented as a lookup table. Each state-action pair in such representation is associated with only one memory location and each such location is used to store the current estimate of the value associated with that pair. Thus, the above update equation has the form:

$$Q(x, a) = Q(x, a) + \alpha(r + \gamma V(y) - Q(x, a)).$$

3. Learning algorithm

The learning method presented (Fig. 1) is based on the above-described Q-learning algorithm (Watkins, 1989). As we already know, it requires learning of only one function, assigning to each state-action pair (x, a) a utility of performing action a in state x, called *action utility* or *Q-value*.

Our main idea is to maintain a set of instances representing historical information. This is a memory of past experiences, where each instance s_j has a structure $s_j = (x_j, a_j, Q_j)$ (i.e. state-action pair and current estimate of its Q-value). We use these instances to generate all possible decision rules according to the rough set method of Skowron (1995) based on discernibility matrix. Another alternative is to generate rules covering the decision classes, and not all rules. These methods are have been implemented, e.g., in LERS (Grzymała-Busse, 1992) and ROSETTA systems (Ohrn and Komorowski, 1997).

Since we generate all rules, many rules can be derived from a single instance and a single rule can be obtained from many instances. Instead of generating rules every time a new instance is added to memory, the agent does it every P step of simulation run. This limits its computational load and prevents it from spending most of the time on generating sets of new rules slightly different from old ones.

With each rule r_i a real value u^i called *utility of rule* is associated, depending on Q-value of instances in memory, from which the rule was derived. Utility of each new rule is computed as a minimum of the Q-values of all instances from which the rule was obtained (step 5). The minimal value was chosen to avoid overestimating initial utilities, which can slow down learning process.

When the agent is about to choose an action it finds a set of rules matched

Let:

$M(x,a) \equiv \{r_i \mid MATCHES(r_i,x,a)\}$, $m = |M(x,a)|$

$MEM \equiv \{s_i \mid s_i \equiv \langle x_i, a_i, Q_i\rangle\}$

Set parameters: α, γ, $INIT\_VAL$

$R_0 := \{r_i \mid r_i$ is the most general rule$\}$

$L := \emptyset$, $MEM := \emptyset$

At each time step t:

1. Observe current state x_t
2. Select an action a_t for state x_t using the action utilities $Q(x_t,a)$ for each a established by the following rule:

   if $|M(x_t,a)| > 0$ **then**

(a) $Q(x_t,a) := u^k$ where $u^k = \max\{u^i | r_i \in M(x_t,a)\}$

   else

   if $(\exists s_i \in MEM)(x_i = x_t \wedge a_i = a_t)$ **then**

(b) $Q(x_t,a) := Q_i$

   else

(c) $Q(x_t,a) := INIT\_VAL$

   endif

3. Perform action a_t ; observe new state x_{t+1} and immediate reinforcement rnf_t
4. Update utilities according to the following rule:

   if $|M(x_t,a_t)| > 0$ **then**

   for each rule $r_i \in M(x_t,a_t)$:

(a) $$u^i := \begin{cases} u^i + \alpha(rnf_t + \gamma \max_a Q(x_{t+1},a) - u^i) & \text{for } i = k \\ u^i + \alpha(\gamma \max_a Q(x_{t+1},a) - u^i) & \text{otherwise} \end{cases}$$

   endfor

   if $(\neg\exists s_l \in MEM)$ such that $(x_l = x_t \wedge a_l = a_t)$ **then**

   $L := L \cup \{\langle x_t, a_t, u^k\rangle\}$

   endif

   else

(b) $Q(x_t,a_t) := Q(x_t,a_t) + \alpha(rnf_t + \gamma \max_a Q(x_{t+1},a) - Q(x_t,a_t))$

   if $(\neg\exists s_l \in MEM)$ such that $(x_l = x_t \wedge a_l = a_t)$ **then**

   $L := L \cup \{\langle x_t, a_t, Q(x_t,a_t)\rangle\}$

   endif

   endif

5. After every P steps:

   $R_{n+1} := GENERATE\_RULES(R_n, MEM, L)$

   with utility u^j of each new decision rule r_j (e.g. $r_j \in R_{n+1} \wedge r_j \notin R_n$) calculated according to the following rule:

   $u^j \equiv \min\{Q_i | s_i \in MEM \cup L \wedge MATCHES(r_j, x_i, a_i)\}$

   Then:

   $MEM := MEM \cup L$, $L := \emptyset$

(step 2a). Using the individual rules utilities the agent calculates Q-value of each action in the current situation as a maximum from the m matched rules:

$$Q(x_t, a) = \max\{u_i \mid MATCHES(r_i, x_t, a)\}.$$

Since, according to the Q-learning algorithm (the current decision policy, to be exact), the agent chooses the action with the highest Q-value, only one rule r_k must be selected, which suggests the subsequent action. If there are more rules matched with the same action and identical utility one is chosen arbitrary. There is also some random exploration based on a normal distribution, which is used during action selection.

As stated above, we generate rules after every P steps. Therefore, there may be quite new—with regard to random exploration we use—state-action pairs, which do not match any rule (steps 2b and 4b). In this situation, each new instance added into memory is treated as a lookup table element (Watkins, 1989), as long as new rules will be derived from it. Therefore, its Q-value calculations and modifications are performed on a single table element only (Step 4b).

After the agent receives reinforcement (step 3), Q-learning modification of utilities is performed. Utilities of those instances in memory, for which decision rules have not been generated yet, are modified according to the standard update rule (step 4b) (Watkins and Dayan, 1992). The modification of decision rule utilities, however, refers only to those of the m matched rules, which suggested the same action as the chosen rule r_k (step 4a). The modification is implemented as the update of individual utilities u^i for all rules matched by current state-action pair as prescribed by the following rule:

$$u^i = \begin{cases} u^i + \alpha(rnf_t + \gamma \max_a Q(x_{t+1}, a) - u^i) & \text{for } i = k \\ u^i + \alpha(\gamma \max_a Q(x_{t+1}, a) - u^i) & \text{otherwise} \end{cases}$$

where α is a learning rate, rnf_t is received reinforcement and γ is a discount factor. The utilities of the other rules are not updated.

4. Simulation results

The cart-pole balancing problem, also called the inverted pendulum problem, is the control problem we decided to use in order to present the performance of our method. The pole balancing task has been studied by Burto, Sutton and Anderson (Barto et al., 1983) and others. The task involves a wheeled cart on a track, with a pole hinged to the top of the cart. At each time step (0.02 second interval) the controller (or agent) must decide whether the cart should apply a force to the left or to the right, in order to keep the pole balanced vertically. The trial is judged a failure when the pole falls too far (≥ 12 degrees) to either the left or the right, or when the cart falls off the track (cart position, in meters,

four system state variables (the position of the cart on the track, the velocity of the cart, the angle of the pole with the vertical, the angular velocity of the pole) and a reinforcement signal of -1 when a trial fails. The output of the controller is a binary value indicating a push on the cart to the left or to the right. The pole balancing is a relatively hard problem with a long reinforcement delay, because the agent receives non-zero reinforcement only at the end of each trial, i.e. after a failure. Even at the beginning of learning, with a very poor policy, a trial may continue for hundreds of time steps, and there may be many steps between a bad action and the resulting failure.

We implemented the pole and cart dynamics according to the equations given in Barto et al. (1983). In order to deal with the continuous state space of cart-pole system we divided it into disjoint regions. The quantization thresholds are also the same as used in Barto et al. (1983) and yield 162 regions. The values of the Q-learning parameters were as follows: the learning rate α was set to 0.50 and the discount factor γ was set to 0.999. We also replicated exploration strategy used during action selection from Barto et al. (1983), and set the value of β to 0.01. The generation period P for rule generation was roughly optimized by a small number of preliminary runs and equal $P = 500$. Action utilities of new state-action pairs were initialized at 0.00. Our approach was compared to the standard version of Q-learning with lookup table function approximation. The values of learning parameters were all the same as for our method.

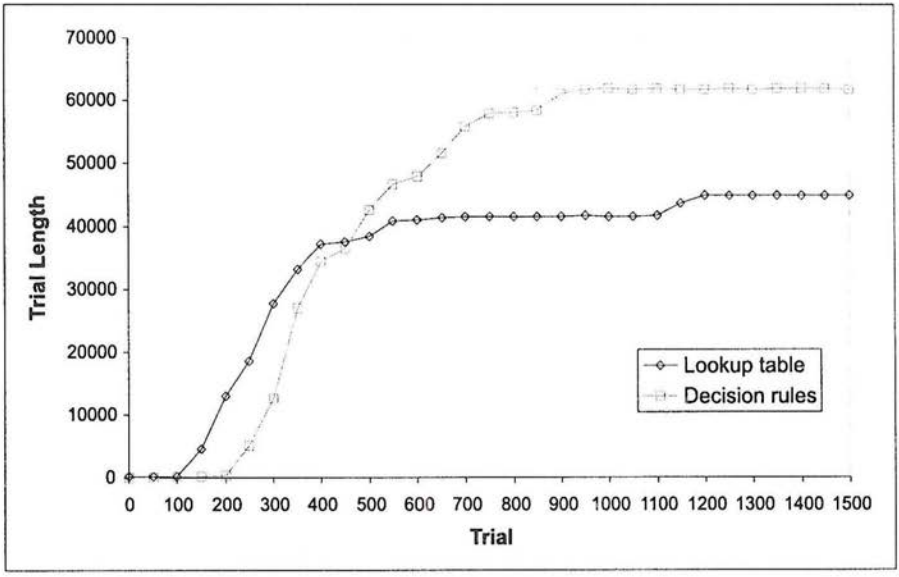

Figure 2. Learning curves for Q-learning with lookup table and decision rules

Our experiments consisted of 30 runs with different random initialization seeds. Each run continued for 1500 trials. The criterion for a successful run was learning to keep the pole balanced for a trial of 100,000 steps. Therefore, some of individual runs were terminated before completing 1500 trials. To produce reliable averages for all 1500 trials, fictitious remaining trials were added to such run, with the duration equal to the last, after which the run was terminated. The results are plotted as the average duration (number of time steps) of the previous 50 consecutive trials versus the trial number.

The results obtained are presented in Fig. 2. As we see, our method (with decision rules) achieved a much better performance level when compared to the standard Q-learning algorithm with lookup table representation, as to the quality of the final policy, and slightly worse level, as to the learning speed. It seems that there are two reasons for slower learning. First, we must take into account that the initial phase of our algorithm required gathering of data for the first rule generation. In the experiment presented each run started out with completely general set of rules to cover all possible states. Thus, the system always makes random moves at first. Second, there is a difficulty in proper action selection, which stems from the number of rules generated. Since all possible rules are generated, it takes too much time to establish optimal utilities for them during later stages of learning so as to differentiate between redundant and essential rules.

5. Concluding remarks

First, we should emphasize that presented algorithm is probably not the only combination of the rough set method of rule generation and the reinforcement learning. We feel, however, that the results achieved are very promising, even if there are several deficiencies. The main one is that the algorithm presented generates all rules without any selection mechanism. Since we use the rough set method, the number of rules can be huge and computational requirements high in general case. Thus, we need another method of rule generation or we should modify the method used. Scope classification, for instance, presented in Lachiche and Marquis (1998) would provide one possible solution to this problem. There are also other modifications of the presented algorithm, which should be considered. For example, using the Variable Precision Rough Set model (VPRS) (Ziarko, 1993) instead of standard rough sets would enable us to generate imprecise rules. Given such representation, better adaptation to the underlying state-action space might be achieved. These and other improvements will be the subject for future work.

Acknowledgements. This work was supported by the grant 8T11A 020 016

References

BARTO, A.G., SUTTON, R.S. and ANDERSON, C.W. (1983) Neurolike elements that can solve difficult learning control problems. *IEEE SMC*, **13**, 835–846.

BRADTKE, S.J. (1993) Reinforcement learning applied to linear quadratic regulation. In: *Advances in Neural Information Processing Systems 5*. Morgan Kaufmann, Los Altos.

GRZYMAŁA-BUSSE, W. (1992) LERS—A system for learning from examples based on rough sets. In: Słowinski, R., ed., *Intelligent Decision Support. Handbook of Applications and Advances of the Rough Set Theory*. Kluwer Academic Publishers, Dordrecht, 3–18.

LACHICHE, N. and MARQUIS, P. (1998) Scope Classification: An Instance-Based Learning Algorithm with a Rule-Based Characterisation. In: *Proceedings of the 10th European Conference on Machine Learning*, Chemnitz, Germany. *LNAI* **1398**, Springer-Verlag.

MCCALLUM, R.A. (1996) Hidden state and reinforcement learning with instance-based state identification. *IEEE Tans. on System, Man, and Cybernetics*, **26** (3) (*Special Issue on Learning Autonomous Robots*).

OHRN, A. and KOMOROWSKI, J. (1997) Rosetta—A Rough Set Toolkit for Analysis of Data. In: *Proc. Third International Joint Conference on Information Sciences*, Durham, NC, USA, March 1–5, Vol. 3, 403–407.

SANTAMARIA, J.C., SUTTON, R.S. and RAM, A. (1996) Experiments with Reinforcement Learning in Problems with Continuous State and Action Spaces. COINS Technical Report 96-088. University of Massachusetts, Amherst.

SKOWRON, A. (1995) Extracting Laws from Decision Tables: A Rough Set Approach. *Computational Intelligence*, **11** (2), 371–388.

SUTTON, R.S. and BARTO, A.G. (1998) Reinforcement Learning: An Introduction. MIT Press, Cambridge.

WATKINS, C.J.C.H. (1989) *Learning from Delayed Rewards*. PhD thesis, Kings College, Univ. of Cambridge, England.

WATKINS, C.J.C.H. and DAYAN, P. (1992) Technical Note: Q-learning. *Machine Learning*, **8**, 279–292.

ZIARKO, W. (1993) Variable Precision Rough Set Model. *Journal of Computer and System Sciences*, **46** (1), 39–59.