Computer Science Journal of Moldova, vol.**10**, no.2(29), 2002

The Analysis of Experimental Results of Reinforcement Learning Systems

Jaroslav E. Poliscuk

Abstract

In this article a reinforcement learning method is analyzed, in which a subject of learning is defined. The essence of this method is the selection of activities by a try and fail process and awarding deferred rewards. If an environment is characterized by the Markov property, then step-by-step dynamics will enable forecasting of subsequent conditions and awarding subsequent rewards on the basis of the present known conditions and actions, relatively to the Markov decision making process. The relationship between the present conditions and values and the potential future conditions are defined by the Bellman equation. Also, the article discussed a method of temporal difference learning, mechanism of eligibility traces, as well as theirs algorithms TD(0) and TD(Lambda). Theoretical analysis were supplemented by the practical studies, with reference to implementation of the Sarsa(Lambda) algorithm, with replacing eligibility traces and the Epsilon greedy policy.

Keywords: Algorithm TD(0), algorithm TD(Lambda), Bellman equation, Markov decision making process, mechanism of eligibility traces, method of temporal difference learning, reinforcement learning method.

1 Introduction

Only intelligent systems have the possibility of adapting to new, unknown and rapidly changing situations. Namely, intelligence is the

feature of systems capable of adapting to changes in environment. Consequently, the more marked the feature, the greater is the intelligence of the system.

System features improve through the process of adaptation. This process takes place through a few different features of intelligence, such as dialog imitation, the solution of all problems variants, non-trivial task solving, extrapolation and learning.

The computer is the only man-made machine with the potential to acquire some of the characteristics of intelligent systems. Sociological, cultural, psychological and other aspects, not directly related to the technical assumptions necessary to achieve this aim, will not be dealt with in this article.

The main characteristics of such artificial intelligence should be: data acquisition, data storage and data processing rate, efficiency and changeability of computer programmes, learning possibilities, extrapolation and non-trivial tasks solving.

The intelligent systems should be placed in a very complex environment which can hardly if at all be defined and modelled. For example: car driving, flight control, supervision of very complicated technological process, etc. In these situations standard programming methods yield only approximate, partial, and, quite often, inadequate results. Machine learning should make it possible to take a different approach to problem solving, i. e. it should provide the system with the tools to learn and solve a problem. The system is expected to be capable to solve the problem routinely at a later stage.

A computer programme or a machine or a system can be considered capable of learning if they can enhance their features by gaining experience in problem solving for given environment. The enhancement of system features defines the processes of knowledge acquisition and knowledge extension. The system should be capable to get new knowledge and to use it more efficiently. Moreover, the system has to be robust enough and so defined as to be able to accept environmental dynamic changes. These requirements could be fulfilled only in the systems whose learning processes are based on interaction between the student and the environment.

2 System environment

In this paper the reinforcement learning method [1],[2],[3],[7],[8], based on learning through the interaction determined by the student's given goal, is analyzed. This method differs from the classical supervised learning methods where the student has been explicitly told what to do for given environment.

The point of reinforcement learning method lies in the idea of involving the deferred awarding of rewards to the student. The student receives the signal of the award every time when he/she manages to put the system in an appropriate state. In this way the learning process is strengthened. There is no predefined task list because it is on the student to find out which actions will bring him/her the greatest reward. In the most challenging cases the actions which have been already taken bring immediate reward but also affect future actions. According to the above mentioned, the essence of this method is the selection of activities by the trial and error process and awarding of deferred awards.

The reinforcement learning method defines the subject of learning rather then the algorithm of learning. This means that any algorithm made to solve a given task will be the algorithm of reinforcement learning method. The *Agent* ('agent' is a much broader term than 'student' because it has to learn and make decisions simultaneously) should take into consideration the most important aspects of the real problem facing them during the process of accomplishing the given goal.

The Agent must do the following:

- find out *the state* of the environment,
- take *actions* to affect the given state,
- have the given *goal or goals* which are closely connected to the state of the environment.

Besides the agent and the environment, four subelements of reinforcement learning method can also be defined as follows [3],[7]:

- *policy*, which translates the state of the environment into an action to be done within that environment, and, which determines behavior of the agent,

- *reward function*, which defines the goal that the agent has to achieve. This function transforms the current state of environment into a scalar, i.e. a reward, which gives to the agent the feedback on the validity of that state,

- *value function*, which defines the long-term validity of the particular state of the environment, also taking into account the states which follow that environment and possible rewards in those states, and, optional.

- *model of the environment*, which purpose is to reflect the environment. This model makes it possible to plan the process, i.e., to make decisions on which action has to be taken into account depending on the possible future states before they are actually reached.

The agent is in permanent communication with the environment, as shown in figure 1. The task, which is an instance of the reinforcement learning method's problem, is defined by the complete specification of the environment.

The agent and its environment communicate one to another in discrete parts of time $t = 0, 1, 2, ...$ At any time t the agent receives the information on the state of the environment, $s_t \in S$, where S represents the assembly of all possible states of environment. The appropriate action, $a_t \in A(s_t)$, is chosen in the same way, where $A(s_t)$ represents the assembly of all possible actions at the particular time. In the next time moment $t+1$, the environment changes its state into s_{t+1}, as the consequence of the action which has already been taken, and sends the scalar reward $r_{t+1} \in R$ to the agent.

At any step of the procedure the agent will choose the action on the basis of the policy π_t, where $\pi_t(s, a)$ represents the probability that for the state of environment $s_t = s$, there is an action $a_t = a$. Algorithms

of the reinforcement learning method show how the agent changes its policy in the course of the process of gaining experience. It is important to emphasize that main goal of the agent is to maximize given rewards in a long-term period.

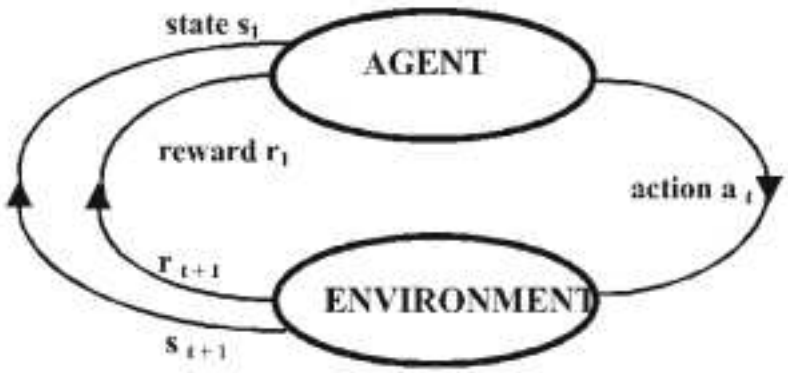

Figure 1. The interaction between the agent and its environment

The boundary between the agent and its environment can be established according to the general rule that everything the agent cannot affect directly makes part of its environment. In practice this boundary can be established after states, actions and rewards are determined, i.e., after the definition of the task.

3 Rewarding and independent way method

During its activity, the agent tends to maximize the total of the received rewards. The reward is the scalar whose value varies for every step of the interaction between the agent and the environment. The above mentioned definition of the reward determines the place of its calculation, and that is the agent‘s environment. Besides, the agent should not by any means influence the process of reward calculation.

The assumption is that the agent has received the following sequence of rewards after the time t: r_{t+1}, r_{t+2}, r_{t+3}, ... and it wants to maximize the total expected reward R_t. The total expected reward depends on the above mentioned sequence, and, the simplest, it is just a sum of the members of the sequence:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \tag{1}$$

where T represents the last time period.

This approach is meaningful only in case of problems where the last time period is defined, i.e. when the interaction between the agent and the environment can be divided into parts like in games. Each part has own end and after this it returns into the beginning state. The tasks which have the parts in the above-mentioned sense are called *part tasks*.

The second type of tasks is the *continual tasks*. The control of processes is an example of this type of tasks. In this case, the expected total reward function can be calculated by using the concept of *discount factor* in the following way:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2}$$

where γ represents the discount factor and $0 \leq \gamma \leq 1$.

Discount factor determines the value of future rewards as it is now , i.e. the reward obtained in k step will have the value of γ^{k+1} times lower than in the present.

The environment informs the agent about its state in every moment. On the basis of that information, the agent makes necessary decisions. The ideal state signal should provide information on both previous and present states, without diminishing the agent‘s prediction feature. It can be said that this signal has *Markov property* or the *independent way method*, because all relevant data are included into the information about the present signal state.

The answer to the question on how the environment in the moment $t+1$ will respond to the action taken in the moment t can be, in general, that the response depends on all what has happened before the $t + 1$ moment.

On the basis of the above mentioned the following conclusion can be drawn: the state has the Markov property if and only if the probability of the transition from one state into another [3],[4],[7] is the following:

$$\begin{aligned} \Pr ob.\{s_{t+1} &= s`, r_{t+1} = r|s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0| \\ &= \Pr ob.\{s_{t+1} = s`, r_{t+1} = r|s_t, a_t|\} \end{aligned} \quad (3)$$

for each $s`, r$ and for all possible values of previous events.

If the environment, i.e. its state, has the Markov property, then such one-step dynamics enables the prediction of the next state and the next reward on the basis of the already known present state and action.

The reinforcement learning method, which satisfies the Markov property, is named *Markov decision process* [4],[7]. In case the spaces of the states and actions are definite and determined with the one step dynamics of the given environment, this type of the Markov decision process (MDP) is called the definite *Markov decision process.*

For an arbitrary given state s and action a, the probability for each possible following state $s`$ is given as:

$$P^a_{ss`} = \Pr ob.\{s_{t+1} = s`|s_t = s, a_t = a\} \quad (4)$$

Also, for the given present state s and action a, together with any next state $s`$, the expected value E of the reward is represented as:

$$R^a_{ss`} = E\{r_{t+1}|s_t = s, a_t = a, s_{t+1} = s`\} \quad (5)$$

These two values $P^a_{ss`}$ and $R^a_{ss`}$ respectively, determine the most important dynamic features of the definite MDP.

4 State value functions and actions and their optimization

Value functions are defined on the basis of the policy followed by the agent. The policy π represents the transformation of the state $s \in S$

and the action $a \in A(s)$ into the probability $\pi(s,a)$ of action a sampling when s represents the state of the environment.

In other words, the *state value function* for the *policy* π, $V^{\pi}(s)$ represents the total reward expected by the agent which applies the policy π since the state s.

The state value function $V^{\pi}(s)$ for the MDP is defined as follows:

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \tag{6}$$

where E_{π} represents the expected value of the reward when the agent follows the policy π.

The *action value function for the policy* π is defined in a similar way:

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t | s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\} \tag{7}$$

The main role of value functions is to satisfy the following recurrent relation. For any policy π and any state s, there is a relation between the value of current state s and the value of the possible next state:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left\{R_t | s_t = s\right\} \\ &= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \\ &= E_{\pi}\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\} \\ &= \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s'\right\}\right] \\ &= \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s')\right] \end{aligned} \tag{8}$$

The above equation is called the *Bellman equation* for the state value function [2]. It represents the relation between the state value and the successor state value. This can be illustrated by means of *"backup"* diagrams.

The main role of these diagrams is to return the information on the successor state value at a particular time. The Bellman equation does the averaging of all possibilities, pondering each possibility with its probability.

The policy π is equal or better than policy π‘ if its total expected reward is equal or greater, i.e., $\pi \geq \pi$‘ if and only if the $V^{\pi}(s) \geq V^{\pi}$‘(s) for every $s \in S$. There is always at least one policy which satisfies this condition. Such policy is called the *optimum policy* π^*. Following this definition it can be concluded that more than one optimum policy can exist at the same time, but all of them share only one *optimum state value function* V^*:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{9}$$

for every $s \in S$.

The same can be applied to the optimum action value Q^*:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{10}$$

for every $s \in S$ and $a \in A(s)$.

For each *state – action (s, a)* pair the above mentioned function gives the total expected reward for the processing of the action a in the state s and allows to apply the optimum policy later on. For this reason the optimum action value function can be defined by using the optimum state value function:

$$Q^*(s,a) = E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = a, a_t = a\} \tag{11}$$

The Bellman equations for the optimum value function are as follows:

$$V^*(s) = \max_{a \in A(s)} \sum P^a_{ss'} [R^a_{ss'} + \gamma V^*(s')] \tag{12}$$

$$Q^*(s) = \sum P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a'} Q^*(s', a') \right] \tag{13}$$

These equations have a unique solution which is independent of the policy. The advantage of these equations is in the fact that when V^* and Q^* are obtained, the process of policy determination is much simpler.

It is very useful to analyze the basic algorithms that make decisions as well as the concepts allowing such analysis. The starting points are the results obtained through practice, without taking into account the theoretical disputes on the convergence of known methods and their countability.

The calculation of the optimum state functions is performed on the basis of Bellman optimum equations (12) and (13). In this case there are two fundamental constraint factors:

- the necessary time factor and
- the necessary space factor.

The necessary time factor is related to the time necessary for the calculation process, and the necessary space factor is related to the memory space necessary for the calculation process. If n is the number of possible states and m is the number of possible actions in these states, then Bellman equations define the system of nm non-linear equations with nm unknown variables. In complex problems the numbers n and m can be very large and their solution by using some of the known methods is unacceptable. For all these reasons the approximate solution method is applied.

The iteration methods using the reinforcement learning method are called the *temporal difference learning methods (TD methods)* [6]. These methods are acceptable in solving of the time factor problem, but they still use the backup mechanism which involves the space factor problem.

All iteration methods for policy definition consist of two iterative simultaneous processes:

- the first process does the value function iteration consistently with the momentary policy – policy evaluation,
- the second process makes the policy become greedy in relation to the momentary value function – policy improvement.

The policy is greedy if:

$$\pi`(s) = \arg_a \max Q^{\pi}(s,a) \tag{14}$$

where $\arg_a \max$ denotes that this term represents the maximum for the action a.

The agent permanently encounters the problem of how to define the optimum policy on the basis of the available state, i.e. how to pick the optimum action from the assembly of possible actions.

The first method is a simple sampling of the action of the greatest value. The action value is evaluated by the iteration process. This method is termed as *'greedy'* because it follows the greedy policy and it has the property of exclusive exploitation because it does not bring any new knowledge.

The above mentioned method can attain a certain level of research by a slight intervention. Namely, with the small specific probability ε, a random action will be selected instead of the best action. This method is used very often in practice and it is called ε - *greedy method.* The selection is conducted according to the rule of uniform probability distribution which sometimes can lead to unexpected results.

The above mentioned disadvantages can be resolved by applying a different probability distribution rule for actions selection. The methods using the different distribution rule are named *softmax methods* and are mostly based on the Bolcman's distribution rule:

$$\Pr ob.(s,a^*) = \frac{e^{Q(s,a^*)/T}}{\sum_{a \in A} e^{Q(s,a)/T}} \tag{15}$$

where the parameter $T > 0$ called *temperature* indirectly controls the level of researching. When $T \to 0$ softmax method is equal to greedy

method, research becomes dominant for high temperature values. Sophisticated heuristic algorithms are much more complex and can influence the changes in the architecture of learning [5].

5 Algorithms of temporal difference methods

The main feature of the temporal difference learning method is the iterativeness of the value function evaluation process. The iterative method follows the idea of the generalization of policy iteration and it consists of two parts. In the first part the initialization of the value function based on arbitrary estimated values is performed, while in the second part the approximate value approaches the real value after updating the estimated values using error signal. This can be written as follows:

New Estimation $\leftarrow$ *Old Estimation + step* $\star$ *(Goal - Old Estimation)*

where *(Goal - Old Estimation)* is the estimation error.

The rule for updating the values is derived from Bellman's equations. TD(0) is the simplest TD algorithm and its updating rule is [7]:

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right] \tag{16}$$

The term $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ is called TD error.

The algorithm in the pseudo code is as follows:

$INITIALIZE\ V(s)$,π
$REPEAT$ $(*\ for\ each\ episode*)$
$INITIALIZE\ s$
$REPEAT$ $(*\ for\ each\ step\ in\ episode*)$
$a \leftarrow action\ on\ the\ basis\ of\ the\ policy\ \pi\ for\ s$
$PERFORMED\ ACTION\ a, OBSERVE\ REWARD\ r$ *and next state* s‘
$V(s) \leftarrow V(s) + \alpha[r + \gamma(s\text{‘}) - V(s)]$
$s \rightarrow s$‘

TO THE LAST STATE s
END

Generally speaking, there are two versions of temporal difference method [1],[3],[7]. The first version of the method improves the policy it applies in action selection. The most significant form of this version is Sarsa algorithm. The second version of the temporal difference method applies one policy responsible for action selection, which is most often greedy policy, but improves the policy which succeeds it. Q algorithm is a representative of this version.

Sarsa algorithm applies the action value function *Q(s,a)* rather then the state value function *V(s)* in the process of iteration. Yet, the rule of updating is very similar to the previous one with regard to the expression (11) and it is as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \qquad (17)$$

The algorithm is:

INITIALIZE $Q(s, a)$
REPEAT $(*$ *for each episode*$*)$
INITIALIZE s
$a \leftarrow$ *action on the basis of the policy* π *for s*
REPEAT $(*$ *for each step in episode* $*)$
PERFORMED ACTION a, TO OBSERVE
REWARD r and next state s‘
$a' \leftarrow$ *action on the basis of the policy* π *for s‘*
$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
$s \leftarrow s', a \leftarrow a'$
TO THE LAST STATE s
END

The Sarsa algorithm converges with the probability value 1 to optimum policy and optimum action value function if all *state - action* pairs are visited for an infinite number of times and if the policy converges to greedy policy. This can be achieved by using ε - *greedy* policy where $\varepsilon = 1/t$.

The implementation of $Sarsa(\lambda)$ algorithm, where $0 \leq \lambda \leq 1$ is the decay trace parameter, with the replacing eligibility traces and ε greedy policy for the given labyrinth is as follows:

```
/ * SARSA Start * /
        /* Initialize Q function */
        InitQfunc (Q);
        /* For each episode */
        for (i = 0; i < (Trials); i + +)
        {
            s = InitState (&x, &y);
            InitTraces (e);
            cnt = 0;
            StepsUntilGoal [i] = 0;
            /* Repeat to end or each step in episode */
            while (((x! = GoalX)||(y! = GoalY)) &&(cnt <
MaxSteps))
            {
            /* Select action *)
              a = SelectAction (Q, Epsilon, s);
            /* Performed action and determine next state */
              NextS = NextState(&x,&y,a);
            /* Accept reward */
              r = Reward(x,y,a);
            /* Select next action */
              NextA = SelectAction(Q,Epsilon,NextS);
              /* Update Q*/
              UpdateQfunc(s,a,r,Q,Alpha,NextS,NextA,e,
Gamma,Lambda);
              /* Update state */
              s = NextS;
            cnt++;
                StepsUntilGoal[i]++;
              }
            }
            /* SARSA End */
```

```
        /* To write essential results */
        end(StepsUntilGoal);
    }
```

The *Q algorithm* is certainly a breakthrough in machine reinforcement learning method. It approximates the optimum action value function immediately and independently on followed policy. Its rule of updating is as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t) \right] \quad (18)$$

INITIALIZE $Q(s, a)$
REPEAT (∗ *for each episode* ∗)
    INITIALIZE s
    REPEAT (∗ *for each step in episode* ∗)
        $a \leftarrow$ *action on the basis of the policy* π *for* s
        PERFORMED ACTION a, *OBSERVE REWARD* r *and next state* s‘
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)$
        $s \leftarrow s$‘
    TO THE LAST STATE s
END

All visited *condition - action* pairs have to be updated correctly, which is the only convergence condition for Q algorithm. In fact, this is a requirement in all machine reinforcement learning methods.

6 The activity traces

Activity traces (eligibility traces) [2] represent one of the fundamental mechanisms of machine reinforcement learning methods. The main idea is to provide memory location for each state which has to trace the statistics of its attendance. Every time when the state is attended,

its activity increases greatly and then it falls until the new attempt. The rule of updating the activity traces $e_t(s)$, is:

$$e_t(s) = \begin{array}{ll} \gamma\lambda_{t-1}(s) & if \quad s \neq s_t \\ \gamma\lambda_{t-1}(s) + 1 & if \quad s = s_t \end{array} \tag{19}$$

where $0 \leq \lambda \leq 1$ is the decay trace parameter. The efficiency of the above mentioned algorithms increases with the involvement of activity traces.

The activity traces defined by the rule (19) are called accumulating eligibility traces, and they are different from the replacing eligibility traces which are actually their modification. The rule for the replacing eligibility traces is as follows:

$$e_t(s) = \begin{array}{ll} \gamma\lambda_{t-1}(s) & if \quad s \neq s_t \\ 1 & if \quad s = s_t \end{array} \tag{20}$$

These two kinds of activity traces can be seen in figure 2.

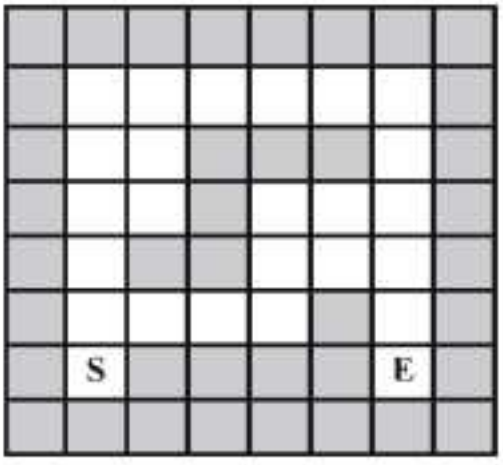

Figure 2. The eligibility traces

The activity traces can be easily implemented in basic TD methods which are then termed as $TR(\lambda)$ *mehtods*. TR(λ) methods represent a generalization of TD methods because in case of $\lambda = 0$, the methods become basic.

The modified updating rules and modified algorithms are explained in the next section.

The *algorithm* $TD(\lambda)$ has the updating rule as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha\delta e(s_t) \tag{21}$$

where $\delta = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ represents TD error.

The algorithm is:

INITIALIZE $V(s)$ *i* $e(s) = 0$ *for each* $s \in S$
REPEAT (* *for each episode* *)
    INITIALIZE s
    REPEAT (* *for each step in episode* *)
        $a \leftarrow$ *action on the basis of the policy* π *for* s
        PERFORMED ACTION a, *OBSERVE REWARD* r *and next state* s‘
        $\delta \leftarrow r + \gamma V(s') - V(s)$
        $e(s) \leftarrow e(s) + 1$
        For each s :
            $V(s) \leftarrow V(s) + \alpha\delta e(s)$
            $e(s) \leftarrow \gamma\lambda e(s)$
            $s \leftarrow s'$
    TO THE LAST STATE s
END

The *Sarsa(λ) algorithm* has the updating rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta e(s_t, a_t) \tag{22}$$

where $\delta = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$ represents TD error.

The algorithm in pseudo code is as follows:

INITIALIZE $Q(s, a)$ *and* $e(s, a) = 0$ *for each* $s \in S$, $a \in A(s)$
REPEAT (* *for each episode* *)

$INITIALIZE\ s$
$a \leftarrow action\ on\ the\ basis\ of\ the\ policy\ \pi\ for\ s$
$REPEAT\ (*\ for\ each\ step\ in\ episode\ *)$
$PERFORMED\ ACTION\ a, OPSERVE\ REWARD\ r$ *and next state* s'
$a' \leftarrow action\ on\ the\ basis\ of\ the\ policy\ \pi\ for\ s'$
$\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
$e(s, a) \leftarrow e(s, a) + 1$
$For\ each\ s,\ a$:
$Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$
$e(s, a) \leftarrow \gamma\lambda e(s, a)$
$s \leftarrow s', a \leftarrow a'$
$TO\ THE\ LAST\ STATE\ s$
END

The $Q(\lambda)$ *algorithm* has the updating rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta e(s_t, a_t) \tag{23}$$

where $\delta = r_{t+1} + \gamma \max Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$ represents TD error.
The algorithm is:

$INITIALIZE\ Q(s, a)\ and\ e(s, a)\ for\ each\ s \in S,\ a \in A(s)$
$REPEAT\ (*\ for\ each\ episode*)$
$INITIALIZE\ s, a$
$REPEAT\ (*\ for\ each\ step\ in\ episode*)$
$a \leftarrow action\ on\ the\ basis\ of\ the\ policy\ \pi\ for\ s$
$PERFORMED\ ACTION\ a, OPSERVE\ REWARD\ r$ *and next state* s'
$a^* = arg_b max\ Q(s', b)$
$\delta \leftarrow r + \gamma Q(s', a^*) - Q(s, a)$
$e(s, a) \leftarrow e(s, a) + 1$
$For\ each\ s, a:$
$Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$
$if\ a' = a^* then\ e(s, a) \leftarrow \gamma\lambda e(s, a)\ else\ e(s, a) \leftarrow 0$
$s \leftarrow s', a \leftarrow a'$

TO THE LAST STATE s
END

On the basis of the above mentioned it can be concluded that estimations of value functions are kept as table data with the entries for each state and each *state - action* pair. In cases of constrained and small number of states and actions this approach gives acceptable results. Otherwise, there is the problem of the lack of memory space. Even if there were infinite memory space, the table problem still remains because of the necessary time search and access to the necessary location.

The solution of the above mentioned problem is in *the generalization* of state and action spaces. The idea is to generalize the small subset of state space and action space through experience in order that they can represent much larger subset of the same space. Even though the idea of generalization is simple, its realization is a huge problem and will be the subject of future researches.

The most frequently used type of generalization in the systems of machine reinforcement learning methods is *function approximation*. This generalization takes particular values of the desired function and tries to generalize them to the degree at which they represent the approximation of that function. The approximation of function represents the instance of learning with supervisor class. The artificial neuron nets, the decision trees, pattern recognition are the members of this class.

7 The analysis of experimental results of machine learning

The above mentioned theories will be illustrated by the results of experiments. For example, the labyrinth problem is one of the trivial problems but still very useful in order to get to the very essence of the examined theories.

The agent task goal is to reach the end of the labyrinth even if it does not know where the end is. In an indirect way, i.e. by means

of the reward, the agent finds out that it has reached the end of the labyrinth. For the purpose of making the example as illustrative as possible as well as to reduce the execution time of the programme, the labyrinth is very simple as can be seen in figure 3.

Once the agent has found the end of the labyrinth (just one episode), it returns to the start position. The system of rewarding for the episode tasks is given in expression (1). But, the method with the reduction factor represented in expression (2) is most frequently applied because for $\gamma = 1$ it becomes the first method.

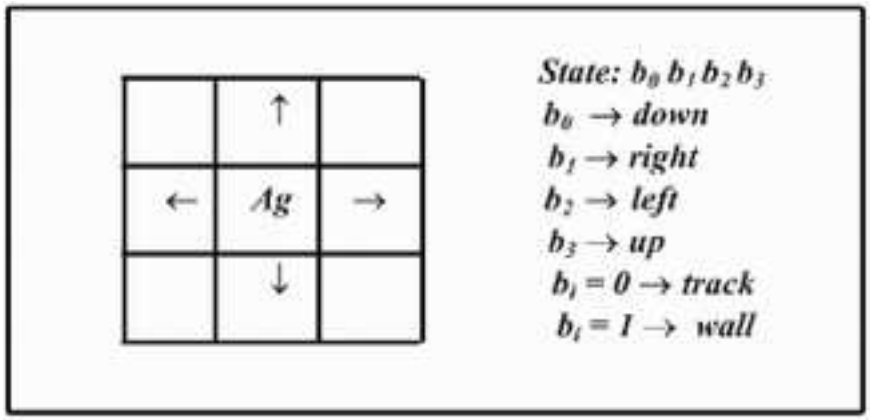

Figure 3. The labyrinth problem

For the implementation of the learning algorithm it is necessary to determine how the state and action will be coded and how the value function and the labyrinth will be stored in the memory.

The labyrinth is stored in the memory as the matrix (8 x 8) of zeroes and ones. Each zero marks the track of the labyrinth and each one marks the wall of the labyrinth. The state is coded with four bits. Each bit represents the position in the labyrinth plane and, also, each bit represents the passable or impassable part of labyrinth track (figure 4). For example, when the agent reaches position 1000, it can continue to the right, left or upwards, as long as the wall is below it. This coding

makes it possible for the number of states to depend on the nature and number of possible actions rather then the size of the labyrinth. There are only four actions: down, right, left and up.

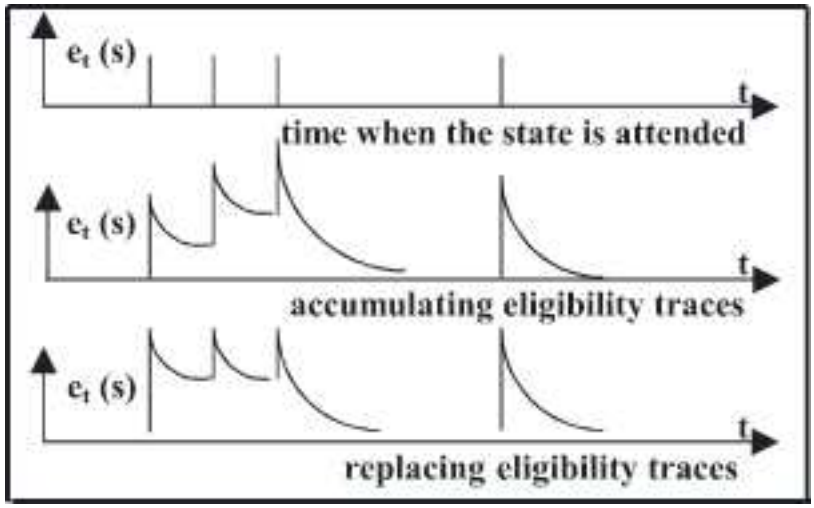

Figure 4. To determine how the state and action will be coded

The value function is stored in memory as the table where the number of rows matches the number of state and the number of columns matches the number of actions.

The implementation of $Sarsa(\lambda)$ algorithm with the replacing eligibility traces and ε greedy policy for the given labyrinth is conducted by using C programming language. The programme consists of about 100 programme instructions and it is used for the necessary experiments. Naturally, the advantage was given to the reinforcement learning method in these experiments. There is a logical sequence inside the programme which generates the file according to episodes are stored. The result given in figure 5 is a graphical representation of the dependance between the number of steps necessary to find the end of labyrinth and the number of episodes.

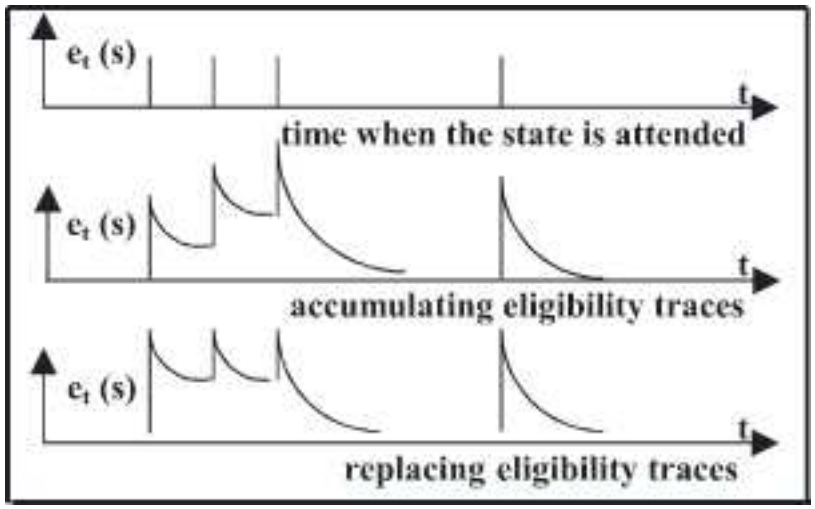

Figure 5. The dependance between the number of steps necessary to find the end of labyrinth and the number of episodes

The implementation eligibility traces:

```
void IntTraces(float e[S][A]
        {
            init i,j;
            for (i = 0; i < S; i++)
                for (j = 0; j < A; j++)
                    e[i][j] = 0.0;
        }
```

and ε greedy policy:

```
        int SelectAction(float Q[S][A], float Epsilon, int s)
        {
            int i;
            int Action;
            int GreedyAction = 0;
            /* Select action with maximum values for state s */
```

```
        for (i = 1; i < A; i++)
            if (Q[s][i] > Q[s][GreedyAction])
                GreedyAction = i;
        /*Select greedy action with probability 1-epsilon %
else select random action */
        if (rand() ¡ Epsilon)
            Action = rand()%A;
        else
            Action = GreedyAction;
        return (Action);
    }
```

The chart gives the pace at which the agent approaches to the optimum policy. The most interesting part of chart are the peaks in particular episodes. These peaks have resulted from the search for the optimum solution using the trial and error and research methods as applied in the machine reinforcement learning method.

Another experiment has also confirmed the assumption that the system performance is much better if activity traces and trace decay parameter λ are involved. As previously mentioned, for $\lambda = 0$ algorithm $Sarsa(\lambda)$ becomes the basic algorithm version. The results of the programme execution for three different values of this parameter are shown in figure 6. What we can see is that system performance is degraded with the decrease in parameter λ. On the basis of the above mentioned it can be concluded that for lower values of λ, the algorithm needs more time for convergence. For $\lambda = 0.65$, the agent cannot find the optimum solution in the first 100 episodes.

8 Conclusions

The method of reinforcement learning method is the most significant form of adaptive machine learning systems. This method became very popular in the last decade due to its thorough theoretical basis, application areas as well as its particular ability to learn without previously prepared knowledge database about the problem. In this method, the

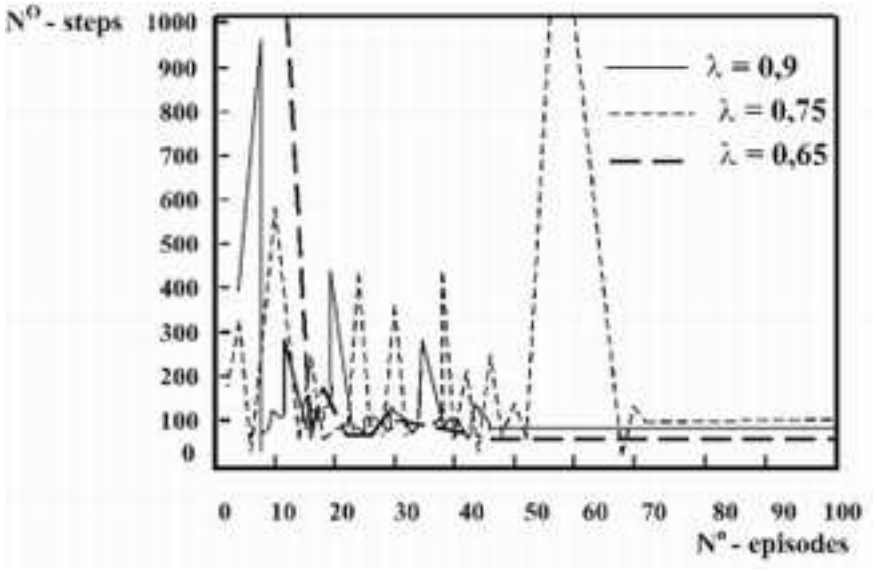

Figure 6. The programme execution for three different values of parameter λ

knowledge database is simply combined with the neuron nets and other methods of supervision.

The main disadvantage of this method is a relatively slow process of learning. This problem can be overcome in several ways. One of the ways is to train the system by using problem simulation and then to apply it in practice. Others use action models, planning models etc. There are also two other problems besides the above mentioned. The first problem is caused by the compromise between the research and exploitation phases, and the second is concerned with the issue of the approximation of the function which determines the size of the task for which this method can be best applied. Particular solutions to the mentioned problems already exist due to which it can be reasonably expected to overcome them in the near future.

Despite the fact that these problems impose considerable limita-

tion, this method has proved to be very useful in all cases where it was possible to apply the system of rewarding. The most important applications are expected to take place in the fields of industrial control, autonomous and mobile robots, as well as in solving optimization problems and resource distribution. This method can also be used for making predictions in various aspects of economy and stock market business.

References

[1] J.A.Boyan, M.L.Littman, *Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach*, Advances in Neural Information Processing Systems: Proceedings of the 994 Conference, San Francisco, CA, USA, 1994.

[2] K.Doya, *Reinforcement Learning in Continuous Time and Space*, Neural Computation, Jan2000, Vol. 12, Issue 1, pp. 219–246.

[3] L.P.Kaelbling, M.L.Littman, A.W.Moore, *Reinforcement Learning: A Survey*, Journal of Artificial Intelligence, Vol. 4, 1996., pages 237–285.

[4] M.E.Lewis, M.L.Puterman, *A Probabilistic Analysis of Bias Optimality in Unichain Markov Decision Process*, IEEE Transactions on Automatic Control, Jan2001, Vol. 46, Issue 1, pp. 96–101.

[5] J.E.Poliscuk, *A contribution to methodology of development of Decision Support Systems and Expert Systems*, Doctors Thesis, Faculty of Organization and Informatics, University of Zagreb, Croatia, 1992.

[6] E.T.Rolls, T.Milward, L.Wiskott, *A Model of Invoviant Object Recognition in the Visual System: Learning Rules, Activation Functions, Lateral Inhibition, and Information - Based Performance Measures*, Neural Computation, Nov2000, Vol. 2, Issue 11, pp. 2547–2573.

[7] R.S.Sutton, A.G.Barto, *Reinforcement Learning: An Introduction*, MIT press - Bradford Books, Cambridge, MA, 1998.

[8] C.Szepesvari, M.L.Littman, *A Unified Analysis of Value - Function – Based Reinforcement – Learning Algorithms*, Neural Computation, 11/15/99, Vol. 11, Issue 8, pp. 2017–2061.

Dr. Jaroslav E. Poliscuk,
Department of Electrical Engineering Podgorica,
University of Montenegro, Yugoslavia
E–mail: *jaroslav@server1.cis.cg.ac.yu*

Received April 12, 2002