STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 1, 2003

A NEW REINFORCEMENT LEARNING ALGORITHM

GABRIELA ŞERBAN

Abstract. The field of Reinforcement Learning, a sub-field of machine learning, represents an important direction for research in Artificial Intelligence, the way for improving an agent's behavior, given a certain feed-back about its performance. In this paper we propose an original algorithm (URU - Utility-Reward-Utility), which is a temporal difference reinforcement learning algorithm. Moreover, we design an Agent for solving a path-finding problem (searching a maze), using the URU algorithm.
Keywords: Reinforcement Learning, Intelligent Agents.

1. Reinforcement Learning

Reinforcement Learning (RL) is the way of improving the behavior of an agent, given a certain feedback about his performance.

Reinforcement Learning [3] is an approach to machine intelligence that combines two disciplines to successfully solve problems that neither discipline can address individually. Dynamic Programming is a field of mathematics that has traditionally been used to solve problems of optimization and control. However, traditional dynamic programming is limited in the size and complexity of the problems it can address.

Supervised learning is a general method for training a parameterized function approximator, such as a neural network, to represent functions. However, supervised learning requires sample input-output pairs from the function to be learned. In other words, supervised learning requires a set of questions with the right answers.

Unfortunately, there are many situations where we do not know the correct answers that supervised learning requires. For these reasons there has been much interest recently in a different approach known as reinforcement learning (RL). Reinforcement learning is not a type of neural network, nor is it an alternative to neural networks. Rather, it is an orthogonal approach that addresses a different, more difficult question. Reinforcement learning combines the fields of dynamic

Received by the editors: December 10, 2002.

2000 *Mathematics Subject Classification.* 68T05.

1998 *CR Categories and Descriptors.* I.2.6[**Computing Methodologies**]: Artificial Intelligence – *Learning*.

programming and supervised learning to yield powerful machine-learning systems. Reinforcement learning appeals to many researchers because of its generality. In RL, the computer is simply given a goal to achieve. The computer then learns how to achieve that goal by trial-and-error interactions with its environment.

A reinforcement learning problem has three fundamental parts [3]:

- *the environment* – represented by "states". Every RL system learns a mapping from situations to actions by trial-and-error interactions with a dynamic environment. This environment must at least be partially observable by the reinforcement learning system;
- *the reinforcement function* – the "goal" of the RL system is defined using the concept of a reinforcement function, which is the exact function of future reinforcements the agent seeks to maximize. In other words, there is a mapping from state/action pairs to reinforcements; after performing an action in a given state the RL agent will receive some reinforcement (reward) in the form of a scalar value. The RL agent learns to perform actions that will maximize the sum of the reinforcements received when starting from some initial state and proceeding to a terminal state. It is the job of the RL system designer to define a reinforcement function that properly defines the goals of the RL agent. Although complex reinforcement functions can be defined, there are at least three noteworthy classes often used to construct reinforcement functions that properly define the desired goals;
- *the value (utility) function* – explains how the agent learns to choose "good" actions, or even how we might measure the utility of an action. Two terms were defined: a policy determines which action should be performed in each state; a policy is a mapping from states to actions. The value of a state is defined as the sum of the reinforcements received when starting in that state and following some fixed policy to a terminal state. The value (utility) function would therefore be the mapping from states to actions that maximizes the sum of the reinforcements when starting in an arbitrary state and performing actions until a terminal state is reached.

In a reinforcement learning problem, the agent receives a feedback, known as *reward* or *reinforcement*; the reward is received at the end, in a terminal state, or in any other state, where the agent has exactly information about what he did well or wrong.

2. A Reinforcement Learning Problem

Let us consider the following problem:
The Problem Definition

We consider an environment represented as a space of states (each state is characterized by its position in the environment - two coordinates specifying the X-coordinate, respectively the Y-coordinate of the current position).

The goal of a robotic agent is to learn to move in the environment from an initial to a final state, on a shortest path (as number of transitions between states).

Notational conventions used in the followings are:

- $M = \{s_1, s_2, ..., s_n\}$ - the environment represented as a space of states;
- $si \in M$, $sf \in M$ - the initial, respectively the final state of the environment (the problem could be generalized for the case of the environments with a set of final states);
- $h : M \to P(M)$ - the transition function between the states, having the following signification: $h(i) = \{j_1, j_2, ..., j_k\}$, if, at a given moment, from the state i the agent could move in one of the states $j_1, j_2, ..., j_k$; we will call a state j that is accessible from state i ($j \in h(i)$) the *neighbor* (*successor*) state of i;
- the transition probabilities between a state i and each neighbor state j of i are the same, $P(i, j) = \frac{1}{card(h(i))}$ (we note with $card(M)$ the number of elements of the set M);

The Goal

The problem will consist in training the agent to find the shortest path to reach the final state sf starting from the initial state si.

For solving this problem, we propose in the followings a reinforcement learning algorithm, based on learning the states' utilities (values), in which the agent receives rewards from interactions with the environment.

3. The URU Algorithm (Utility-Reward-Utility)

The algorithm described in this section is an algorithm for learning the states' values, a variant of learning based on Temporal Differences [1].

The algorithm's idea is the following:

- the agent starts with some initial estimates of the state's utilities;
- during some training episodes, the agent will experiment some paths from si to sf (possible optimal), updating, properly, the states' utilities estimations;
- during the training process the states' utilities estimations converge to the exact values of the states' utilities, thus, at the end of the training process, the estimations will be in the vicinity of the exact values.

We make the following notations:

- $U(i)$ - the estimated utility of the state i;
- $R(i)$ - the reward received by the agent in the state i.

The URU Algorithm

The algorithm is shown in Figure 1.

---

(1) Initialize the state utilities with some initial values;
(2) Initialize the current state with the initial state $sc := si$;
(3) Choose a state s neighbor of sc ($s \in h(sc)$), using some known action selection mechanisms (ϵ-Greedy or SoftMax [2]), following the steps:
  (a) determine the set of successors of the current state ($m = h(sc)$);
  (b) if the current state has no successors (m is empty), return to the previous state ($s := sc$); otherwise go to step (c);
  (c) select from m a subset $m1$ containing the states that were not visited yet in the current training sequence;
  (d) choose a state s from $m1$ using a selection mechanism.
(4) determine the reward r received by the agent in the state sc;
(5) if the current state is not final, then update the utility of the current state as follows:

$$U(sc) := U(sc) + \alpha \cdot (r + \gamma \cdot U(s) - U(sc)) \tag{1}$$

where $\alpha \in (0, 1)$ is a fixed parameter (the *learning rate*), and $\gamma \in (0, 1)$ is a fixed parameter (the *reward factor*).
(6) $sc := s$;
(7) repeat the step 3 until sc is the final state;
(8) repeat the steps 2-7 for a given number of training episodes.

FIGURE 1. The Reward-Utility-Reward (URU) Algorithm.

We have to make the following specifications:

- the training process during an episode has the complexity in the worst case $O(n^2)$, where n is the number of the environment's states;
- in a training sequence, the agent updates the utility of the current state using only the selected successor state, not all the successors (the temporal difference characteristic).

4. Case Study

It is known that the estimated utility of a state [1] in a reinforcement learning process is the estimated *reward-to-go* of the state (the sum of rewards received from the given state to a final state). So, after a reinforcement learning process, the agent learns to execute those transitions that maximize the sum of rewards received on a path from the initial to a final state.

If we consider the reward function as: $r(s) = -1$ if $s \neq sf$, and $r(s) = 0$, otherwise, it is obvious that the goal of the learning process is to minimize the

number of transitions from the initial to the final state (the agent learns to move on the shortest path).

For illustrating the convergence of the algorithm, we will consider that the problem is one of learning the shortest path (the reward function is as we described above), and the environment has the following characteristics:

- the environment has a rectangular form;
- at a given moment, from a given state, the agent could move in four directions: North, South, East, West.

In the followings, we will enounce an original theorem that gives the conditions for convergence of the URU algorithm.

Theorem 1. *Let us consider the learning problem described above and which satisfies the conditions:*

- *the initial values of the states' utilities (the step (1) of the URU algorithm described in Figure 1) are calculated as:* $U(s) = -d(s, sf) - 2$*, for all* $s \in M$*, where* $d(s1, s2)$ *represent the Manhattan distance between the two states;*
- $\gamma \leq \frac{1}{3}$

In this case, the URU algorithm is convergent (the states' utilities are convergent after the training sequence).

The Theorem 1 proving is based on the following lemmas:

Lemma 2. *At the n-th training episode of the agent the following inequalities hold:* $U_n(i) \leq -2$*, for all* $i \in M$*.*

Lemma 3. *At the n-th training episode of the agent the following inequalities hold:* $|U_n(i) - U_n(j)| \leq 1$*, for each transition from* $i (i \neq sf)$ *to* j *made by the agent in the current training sequence.*

Lemma 4. *The inequalities* $U_{n+1}(i) \geq U_n(i)$ *hold for all* $i \in M$ *and for all* $n \in N$*, in other words the states' utilities increase from a training episode to another.*

Theorem 2 gives the equilibrium equation of the states' utilities after applying the URU algorithm.

Theorem 5. *In our learning problem, the equilibrium equation of the states' utilities is given by the following equation:*

$$U^*_{URU}(i) = R(i) + \frac{\gamma}{card(h(i))} \cdot \sum_{j \text{ successor of } i} U^*_{URU}(j) \tag{2}$$

for all $i \in M$*, where* $U^*_{URU}(i)$ *represents the exact utility of the state i, obtained after applying the URU algorithm. We note by* $card(M)$ *the number of elements of the set* M*.*

5. An Agent for Maze Searching

5.1. **General Presentation.** The application is written in JDK 1.4 and implements the behavior of an Intelligent Agent (a robotic agent), whose purpose is coming out from a maze on the shortest path, using the algorithm described in the previous section (URU).

We assume that:

- the maze has a rectangular form; in some positions there are obstacles; the agent starts in a given state and tries to reach a final (goal) state, avoiding the obstacles;
- from a certain position on the maze the agent could move in four directions: north, south, east, west (there are four possible actions);

5.2. **The Agent's Design.** The basis classes used for implementing the agent's behavior are the followings:

- **CState**: defines the structure of a *State* from the environment. This class has methods for:
  - setting components (the current position on the maze, the value of a state, the utility of a state);
  - accessing components;
  - calculating the utility of a state;
  - verifying if the state is accessible (does this contain or not an obstacle).
- **CList**: defines the structure of a list of objects. The main methods of the class are for:
  - adding elements;
  - accessing elements;
  - updating elements.
- **CEnvironment**: defines the structure of the agent's environment (it depends on the concrete problem - in our example the environment is a rectangular maze).
- **CNeighborhood**: the class that defines the accessibility relation between two states of the environment;
- **CRLAgent**: the main class of the application, which implements the agent's behavior and the learning algorithm.
  The private member data of this class are:
  - **m**: the agent's environment (is a *CEnvironment*);
  - **v**: the accessibility relation between the states (is a *CNeighborhood*);

  The public methods of the agent are the followings:
  - **readEnvironment**: reads the information about the environment from an input stream;

- **writeEnvironment**: writes the information about the environment in an output stream;
- **learning**: is the main method of the agent, which implements the URU algorithm; based on this algorithm, the agent updates the utilities of the environment's states.
- **next**: the agent determines the next state where to move (this is made after the learning process took place).

Besides the public methods, the agent has some private methods used in the method **learning**.

5.3. **Experimental Results.** For our experiment, we consider the environment shown in Figure 2. The state marked with 1 represents the initial state of the agent, the state marked with 2 represents the final state and the states filled with black are containing obstacles (which the agent should avoid).

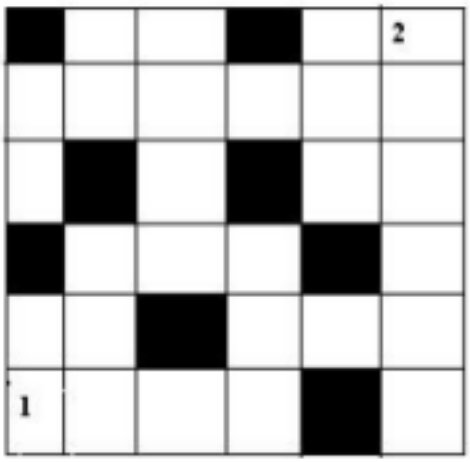

FIGURE 2. The agent's environment

For the environment described in Figure 2, we use the URU algorithm, with the following initial settings:

- $\gamma = 0.9$;
- $\alpha = 0.01$;
- number of episodes = 10;
- as a selection mechanism, we choose the ϵ-Greedy selection, with ϵ=0.1.

The results obtained after the URU learning are presented in Table 1. The states from the environment are numbered from 1 to 36, starting with the corner

TABLE 1. The states' utilities after the training episodes with the URU algorithm

State	Episode 1	Episode 2	Episode 3	Episode 4	Episode 5	Episode 6	Episode 7	Episode 8	Episode 9	Episode 10
1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	-6.0000	-6.0000	-6.0000	-6.0000	-6.0000	-6.0000	-5.9710	-5.9422	-5.9422	-5.9422
3	-5.0000	-5.0000	-5.0000	-5.0000	-5.0000	-5.0000	-4.9780	-4.9561	-4.9561	-4.9561
4	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
5	-3.0000	-2.9760	-2.9760	-2.9760	-2.9522	-2.9522	-2.9522	-2.9522	-2.9522	-2.9522
6	-2.0000	-2.0000	-2.0000	-2.0000	-2.0000	-2.0000	-2.0000	-2.0000	-2.0000	-2.0000
7	-8.0000	-8.0000	-8.0000	-8.0000	-8.0000	-8.0000	-7.4610	-7.1124	-7.1124	-7.1124
8	-7.0000	-7.0000	-7.0000	-7.0000	-7.0000	-7.0000	-6.9640	-6.9267	-6.9267	-6.9267
9	-6.0000	-6.0000	-6.0000	-6.0000	-6.0000	-6.0000	-5.9650	-5.9303	-5.9303	-5.9303
10	-5.0000	-5.0000	-5.0000	-5.0000	-5.0000	-5.0000	-5.0000	-4.9779	-4.9779	-4.9779
11	-4.0000	-3.9790	-3.9790	-3.9790	-3.9581	-3.9581	-3.9581	-3.9436	-3.9436	-3.9436
12	-3.0000	-3.0000	-3.0000	-3.0000	-2.9919	-2.9919	-2.9919	-2.9839	-2.9839	-2.9601
13	-9.0000	-9.0000	-9.0000	-9.0000	-9.0000	-9.0000	-8.3031	-7.8600	-7.8600	-7.8600
14	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
15	-7.0000	-7.0000	-7.0000	-7.0000	-7.0000	-7.0000	-6.9580	-6.9580	-6.9580	-6.9580
16	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
17	-5.0000	-4.9720	-4.9720	-4.9720	-4.9720	-4.9720	-4.9720	-4.9720	-4.9720	-4.9720
18	-4.0000	-3.9850	-3.9850	-3.9850	-3.9641	-3.9641	-3.9641	-3.9435	-3.9435	-3.9230
19	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
20	-8.3664	-8.3166	-7.7394	-7.7394	-7.6956	-7.6956	-7.6521	-7.6521	-7.0875	-7.0498
21	-7.9510	-7.9024	-7.8541	-7.8541	-7.8063	-7.8063	-7.7592	-7.7592	-7.7122	-7.6655
22	-6.9640	-6.9282	-6.8926	-6.8926	-6.8573	-6.8573	-6.8573	-6.8573	-6.8218	-6.7866
23	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
24	-5.0000	-4.9720	-4.9720	-4.9720	-4.9442	-4.9442	-4.9442	-4.9167	-4.9167	-4.8893
25	-10.9300	-10.8584	-10.7872	-9.7732	-9.7732	-9.7105	-9.7105	-9.6483	-9.5865	-9.5233
26	-9.2171	-9.1601	-8.4560	-8.4138	-8.3629	-8.3214	-8.2712	-8.2303	-7.5718	-7.5273
27	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
28	-7.9510	-7.9024	-7.8541	-7.8541	-7.8063	-7.7588	-7.7588	-7.7117	-7.6691	-7.6268
29	-6.9640	-6.9282	-6.8927	-6.8927	-6.8573	-6.8166	-6.8166	-6.7761	-6.7761	-6.7359
30	-6.0000	-5.9650	-5.9650	-5.9650	-5.9303	-5.9017	-5.9017	-5.8675	-5.8675	-5.8058
31	-11.9230	-11.8466	-11.7707	-10.5639	-10.5006	-10.4349	-10.3725	-10.3079	-10.2438	-10.1801
32	-10.9298	-10.9298	-10.8578	-10.7846	-10.7120	-10.6445	-10.5731	-10.5047	-10.4342	-10.4342
33	-9.9430	-9.9430	-9.8864	-9.8864	-9.8864	-9.1123	-9.1123	-9.0559	-9.0068	-9.0068
34	-8.9500	-8.9500	-8.9003	-8.9003	-8.9003	-8.2313	-8.2313	-8.1823	-8.1376	-8.0896
35	-7.9570	-7.9084	-7.8662	-7.8662	-7.8184	-7.7768	-7.7768	-7.7768	-7.7768	-7.7293
36	-7.0000	-6.9580	-6.9580	-6.9580	-6.9163	-6.8806	-6.8806	-6.8806	-6.8806	-6.8393

left-up, in order of the lines. On the columns are presented the estimated values of the states' utilities, during the training episodes.

From Table 1 it is obvious that the states' utilities grow during the training episodes. After the training, the agent will report the learned path (from the initial to the final state), that is the path **(6-1), (5-1), (5-2), (4-2), (4-3), (4-4), (5-4), (5-5), (5-6), (4-6), (3-6), (2-6), (1-6)**. As a policy for moving in the environment after the learning, we consider that, from a given state, the agent will move to a neighboring state that was not visited yet, and having a maximum utility (of course, to determine the policy, we could use some probabilistic action selection mechanisms).

In order to illustrate the experimental results, we will give, in the followings, graphical representations that confirm the theoretical results from the previous sections. Figure 3 presents the change of the initial state's utilities during the training episodes (it is obvious that the utilities grow during the training).

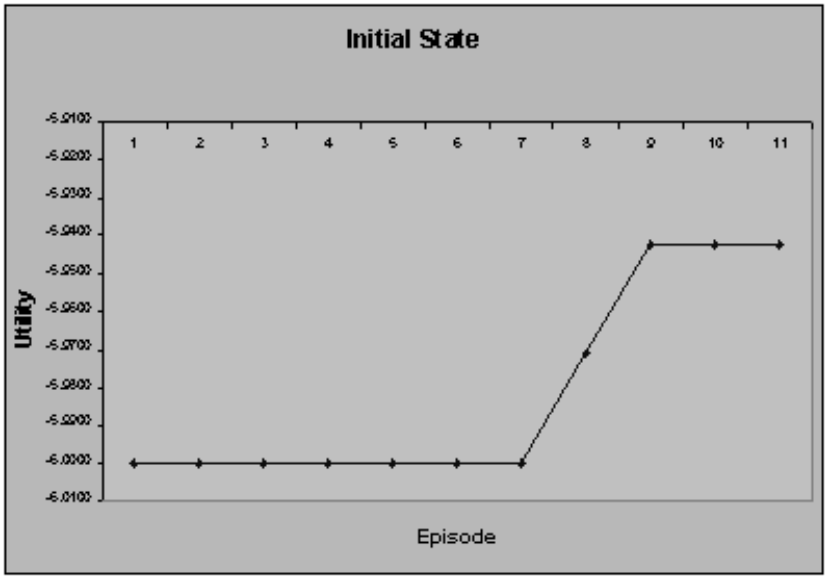

FIGURE 3. The initial state's utilities during the training process

Figure 4 presents the graphical representation of the states' utilities on the first training episode, and Figure 5 presents the graphical representation of the states' utilities on the last training episode. Analyzing comparatively the two figures, we observe that, for each state of the environment, the utilities grow during the training.

In Figure 6 we present comparatively the states' utilities on the first, the 10th and the 5th training episode.

5.4. **Experimental comparison between the URU algorithm and the TD** [1] **(Temporal Difference) algorithm.** In section 4 we illustrate that, in the case of the learning problem in the environment shown in Figure 2, by applying the TD algorithm the states' utilities are not convergent. This fact is presented in Table 2, where the utilities of the first five states during the training are described, results obtained by applying the TD algorithm for our problem.

From Table 2 it is obvious that the states' utilities have not a monotonic behavior, in other words, for some states the utilities increase, for other states the utilities decrease along the training, which does not guarantee the convergence of the algorithm.

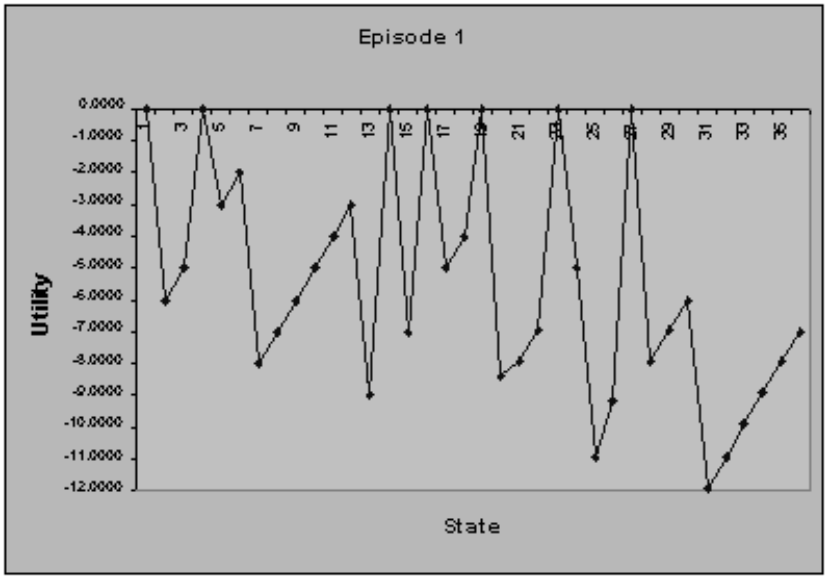

FIGURE 4. The states' utilities on the first training episode

TABLE 2. The states' utilities after the training episodes with the URU algorithm

State	Episode 1	Episode 2	Episode 3	Episode 4	Episode 5	Episode 6	Episode 7	Episode 8	Episode 9	Episode 10
1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	-6.0000	-6.0000	-6.0000	-6.0002	-6.0002	-6.0002	-6.0002	-6.0002	-6.0202	-6.0202
3	-5.0000	-5.0000	-5.0000	-5.0200	-5.0200	-5.0400	-5.0400	-5.0400	-5.0600	-5.0600
4	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
5	-2.9900	-2.9801	-2.9703	-2.9703	-2.9703	-2.9606	-2.9606	-2.9606	-2.9510	-2.9510
...										

6. Conclusions and Further Work

As we have mentioned in the previous sections, the URU algorithm is a variant of a RL algorithm based on temporal differences (if $\gamma = 1$ URU becomes the classical temporal difference learning algorithm - TD).

In comparison with the classical TD algorithm, the following remarks may be considered. These follow naturally from the theoretical results described in Section 4):

- the states' utilities grow faster in the URU algorithm than in TD algorithm, in other words $U_{URU}(i) > U_{TD}(i)$, for all $i \in M$, which means

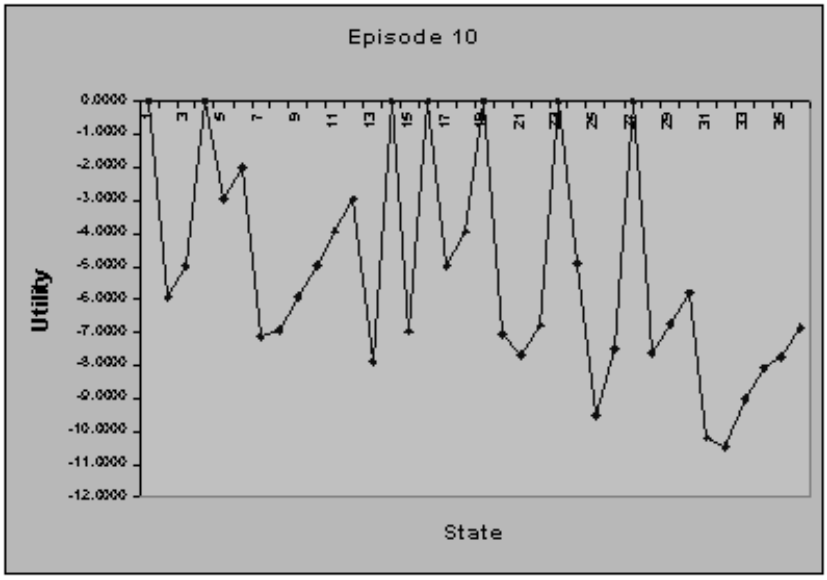

FIGURE 5. The states' utilities on the last training episode

that the URU algorithm converge faster to the solution than the TD algorithm;

- in the case of our learning problem, as we proved in Theorem 1, for $\gamma = 1$ (the TD algorithm), we cannot prove the convergence of the states' utilities.

Further work is planned to be done in the following directions:

- to analyze what happens if the transitions between states are nondeterministic (the environment is a Hidden Markov Model [4]);
- to analyze what happens if the reward factor (γ) is not a fixed parameter, but a function whose values depend on the current state of the agent.
- to develop the algorithm for solving path-finding problems with multiple agents.

REFERENCES

[1] Russell, S.J., Norvig, P.: Artificial Intelligence. A Modern Approach. Prentice-Hall, Englewood Cliffs, NJ, 1995

[2] Sutton, R., Barto, A., G.: Reinforcement Learning. The MIT Press, Cambridge, England, 1998

[3] Harmon, M., Harmon, S.: Reinforcement Learning – A Tutorial, Wright State University, http://www-anw.cs.umass.edu/∼mharmon/rltutorial/frames.html, 2000

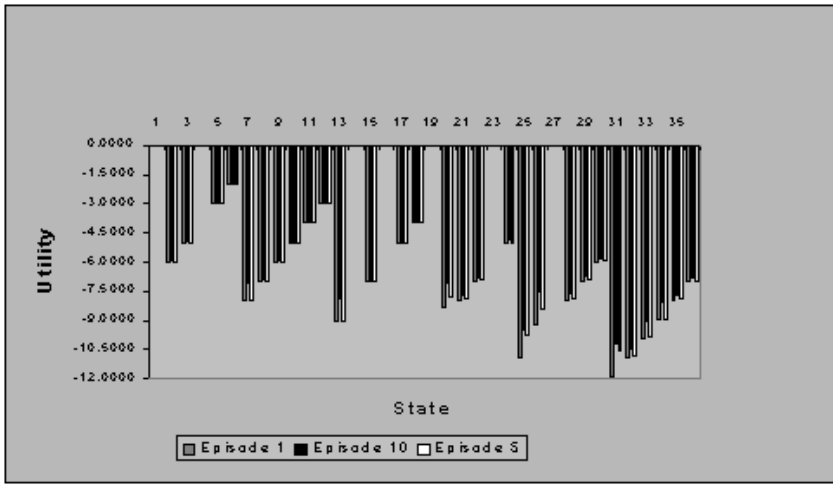

FIGURE 6. The states' utilities on the first, the 10th and the 5th training episode

[4] Serban, G.: Training Hidden Markov Models – A Method for Training Intelligent Agents, Proceedings of the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems, Krakow, Poland, 2001, pp. 267-276

BABEŞ-BOLYAI UNIVERSITY, CLUJ-NAPOCA, ROMANIA
E-mail address: gabis@cs.ubbcluj.ro