Computer Science Journal of Moldova, vol.**12**, no.3(36), 2004

A New Approach in Agent Path-Finding using State Mark Gradients

Florin Leon Mihai Horia Zaharia Dan Gâlea

Abstract

Since searching is one of the most important problem-solving methods, especially in Artificial Intelligence where it is often difficult to devise straightforward solutions, it has been given continuous attention by researchers. In this paper a new algorithm for agent path-finding is presented. Our approach is based on environment marking during exploration. We tested the performances of Q-learning and Learning Real-Time A* algorithm for three proposed mazes and then a comparison was made between our algorithm, two variants of Q-learning and LRTA* algorithm.

Keywords: artificial intelligence, path-finding, maze, reinforcement learning, Q-learning, LRTA*, agents

1 Introduction

A very important problem of distributed computing is to increase the asynchronicity of communication in order to decrease communication costs. Following that direction of thinking, in the last decade the agent paradigm was developed, although the asynchronous approach increased the rate of use for any single or multiple known resource. The basic idea of multiagent systems was to achieve better performance of data search in distributed databases due to its inherent asynchronous work model. At this hour those systems have many different applications e.g. data mining and complex simulations. However, agent-based systems have the same problem inherited from their distributed systems support. This is the problem of finding the path in the network. We propose a new approach in this direction beginning with the idea

©2004 by F. Leon, M.H. Zaharia, D. Gâlea

that we can make the agent act somehow like simple biological beings that can mark the discovered (followed) path in order to later remind it.

A path-finding problem has two main components [8]:

- a set of nodes N, where each node $n \in N$ represents a state;
- a set of directed links L, where each link $l \in L$ is an operator available to the problem solver.

In general, if the problem solver is an agent in a multiagent environment, an operator is an action that can be performed by the agent in the current state. Usually, there is a unique start node $s \in N$, representing the initial state of the agent, and a set of goal nodes $G \subset N$, representing the desired states, i.e. the solutions of the path-finding problem.

Each link has a corresponding weight w_l, which represents the cost of applying the operator (or performing the action). Sometimes, the link weight is called the distance between the two nodes. The nodes that have directed links from a node are its neighbors.

The maze is an example of a path-finding problem in a grid state space. The states can be either free states (where the agent can move) or obstacle states (inaccessible to the agent). The allowed operators are moves along x and y axes (north, east, south, west); diagonal moves are forbidden. Classically, the start node is the upper-left corner and the goal node is the bottom-right corner of the maze. In this paper, we used a more general approach, allowing the goal node to be placed anywhere in the maze.

2 Path-Finding Using Q-Learning

Reinforcement learning [6, 3] is a learning technique based on the maximization of a (usually numerical) reward signal, given as a consequence for taking a certain action in a certain state. The learning agent is not told directly what to do, but it must discover the actions that will give it the highest reward. In some cases, actions may affect not only the

immediate reward, but also the next situation and, through that, all subsequent rewards. These two characteristics, trial and error search and delayed reward, are the two most important distinguishing features of reinforcement learning.

The Q-learning algorithm [7] is a popular reinforcement technique, often used in multiagent learning systems. The main idea of the algorithm is to learn an action-value function, $Q : S \times A \rightarrow R$, that estimates the long-term discounted rewards for each pair state-action. If an action $a \in A$ in state $s \in S$ produces a reward $r \in R$ and a transition to state $s' \in S$, the corresponding Q value is modified as follows:

$$Q(s,a) = Q(s,a) + \alpha \cdot (r + \gamma \cdot \max_{a' \in A} Q(s',a') - Q(s,a)), \tag{1}$$

where α is the learning rate and γ the discount factor. If $\gamma = 0$, the agent is interested only by maximizing its immediate reward. Such a strategy ignores future rewards, so that the total gain (value) may be smaller. The closer γ is to 1, the more important future rewards are for the learning agent.

The optimal policy the agent must use is:

$$\pi^* = \arg\max_a Q(s,a). \tag{2}$$

In our maze problem, the set of actions is:

$$A = \{north, east, south, west\}, \tag{3}$$

and the set of states:

$$S = \{s_{ij}, 1 \leq i \leq mazeheight, 1 \leq j \leq mazewidth\}, \tag{4}$$

$$s_{ij} \in \{free, obstacle, start, stop\}. \tag{5}$$

The rewards we used were:

- 100 for reaching the stop state;

- -100 for trying to move to an obstacle state;
- 0 for moving to another free state.

We first did not place any constraints on the agent movement. Thus, the agent could move to an obstacle state, it would be considered a terminal state and the algorithm would be restarted in a new learning episode. However, in order to speed up the learning, we decided that if the agent chose to move to an obstacle state, it would be given a -100 penalty but then it would keep its former position. This approach is more realistic considering a real situation with a robot moving through a physical maze. When it touches a wall, it doesn't necessarily break, so that the learning should proceed again from the start position.

One of the challenges that arise in reinforcement learning is the tradeoff between exploration and exploitation [6]. To obtain a bigger reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions it has to try actions that it has not selected before. The agent has to exploit what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future.

The initial exploration rate must be high, and then it is gradually decreased each time the agent finds the solution. These values may differ according to the problem [1]. We chose an initial exploration rate $\epsilon_0 \in \{0.8, 0.9, 1\}$ and performed several performance tests. When the agent finds a solution, this rate is decreased by 0.05. However, it cannot become less than 0.1. At the beginning, the agent doesn't know its path (the Q values are initialized at small random values in order not to bias any direction) and it must rely on exploration. As it finds solutions, the exploration rate is decreased down to 0.1. If, after several episodes, the Q function has not converged, the agent must still have a chance to find a solution through exploration.

Two cases were considered: in the first one, the agent simply applies Q-learning, in the second, it marks the states that it has visited and receives a -10 penalty if it goes back again to an already visited state. This is done to discourage the agent to enter any loops.

The performance of the agent is given by the number of learning episodes in which it can learn the path. It must be mentioned that learning the path is not the same thing as finding the path. The agent will find the path many times, but it is important to adjust its Q-values in order to be able to eventually find the solution in one single try, relying only on these Q-values.

We tested the algorithm on three mazes: a small one (11x11) with only one possible solution, a bigger one (25x25) also with only one possible solution, and another big one (19x30), which allowed multiple solutions (see figures 7, 8, and 9). We measured the number of learning episodes and the total steps needed for the agent to reach the solution. A statistical processing of the data was then made, and mean value and standard deviation were computed:

$$m_X = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i \tag{6}$$

$$\sigma_X = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} (x_i - m_X)^2}. \tag{7}$$

Table 1 presents these results for different values of the initial exploration rate ϵ_0, obtained after 20 learning trials:

It must be noted that in both cases, the initial exploration rate has a great influence on the number of learning episodes. When the agent doesn't remember the visited states, learning is the fastest when ϵ_0 is 1 (figure 1).

However, the number of steps in every learning episode is bigger when ϵ_0 is bigger. If the behavior of the agent is driven only by exploration, it performs in a random manner, and so the number of states it goes through is bigger. When ϵ_0 is smaller, we have a greater number of learning episodes, but in each episode the agent finds the solution faster, because it relies more on "real" knowledge (figure 2).

After every successful learning episode, the exploration rate is decreased and the agent begins to use its previously acquired information, therefore the number of steps to reach the solution in each learning

Table 1. The performance of Q-learning with different initial exploration rates in the presence and absence of state marks.

Maze type	**State marks**	ϵ_0	$m_{episodes}$	$\sigma_{episodes}$	$m_{totalsteps}$	$\sigma_{totalsteps}$
small (11x11) single solution	No	1	9.1	3.375	19,727	5,880
		0.9	13.15	4.028	13,516	3,067
		0.8	17.75	7.569	14,814	6,134
	Yes	1	8.9	4.625	21,671	9,627
		0.9	8.25	2.385	8,830	2,203
		0.8	9.85	2.475	5,907	1,573
big (25x25) single solution	No	1	18.55	4.717	1,368,121	285,868
		0.9	191.8	26.658	749,443	60,625
		0.8	295.65	37.936	779,560	137,781
	Yes	1	>1000		~73,900,000	
		0.9	12.6	4.363	106,253	30,783
		0.8	13.45	3.263	57,640	14,753
big (19x30) multiple solutions	No	1	43.7	10.194	138,489	26,510
		0.9	99.55	22.675	106,341	16,196
		0.8	151.3	34.134	98,464	15,771
	Yes	1	101.3	51.424	329,483	169,084
		0.9	68.8	23.134	74,729	19,438
		0.8	59	18.501	41,541	8,003

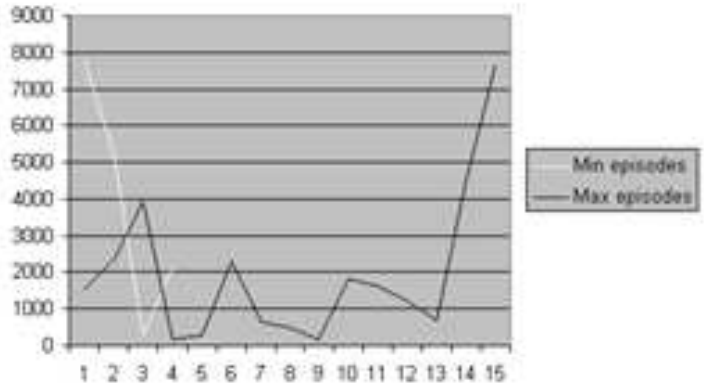

Figure 1. The number of steps per learning episode for a simple single solution maze without state marks and initial exploration rate 1.

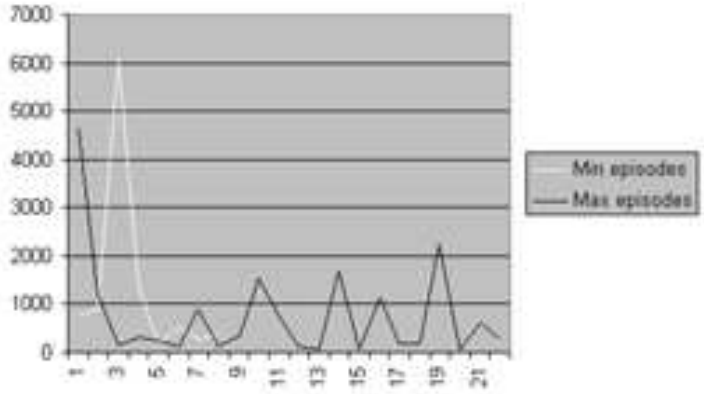

Figure 2. The number of steps per learning episode for a simple single solution maze without state marks and initial exploration rate 0.9.

episode gradually diminishes (figure 3). There are still peaks, because the exploration rate is never null, and a random decision in a key state (e.g. a bifurcation where one way leads to the solution and the other to a dead end) can greatly influence the outcome of the learning process.

A different phenomenon appears when the agent marks the states and it receives a penalty for coming back to an already visited state. In such a case, the best performance is obtained with $\epsilon_0 = 0.9$, rather than $\epsilon_0 = 1$ (figure 4). The randomization of the learning process impedes over the performance of the agent, because the chances are greater to return to a marked state. The agent modifies its Q-values and tries to avoid that state, although it may be a useful one. When the initial exploration rate is less than 1, the algorithm has better performance than in the simple case. Also, when $\epsilon_0 = 0.9$, the number of steps per episode is usually smaller than in the no-marks case.

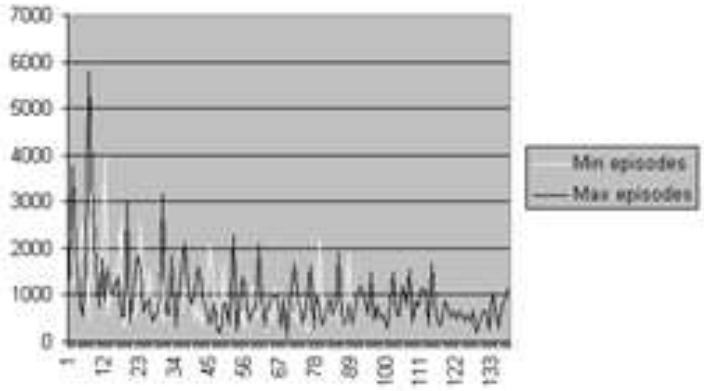

Figure 3. The number of steps per learning episode for a complex single solution maze without state marks and initial exploration rate 0.9.

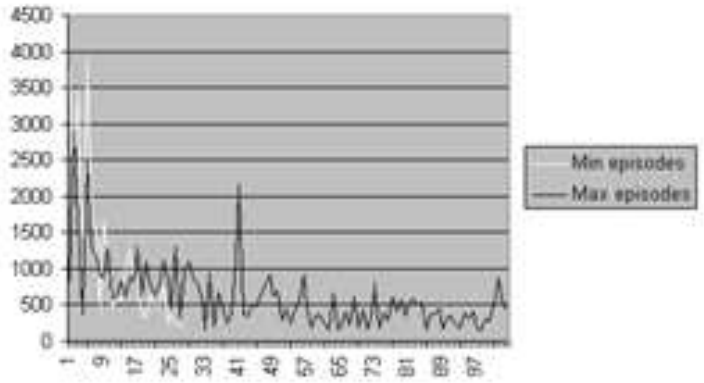

Figure 4. The number of steps per learning episode for a big multiple solution maze with state marks and initial exploration rate 0.8.

3 The Influence of Learning Rate and Discount Factor

After demonstrating that the agent learns quicker if it uses state marks with an initial exploration rate of 0.9, we tried to analyze the influence of learning rate α and discount factor γ (see equation 1) on the path-finding efficiency. Table 2 presents the results for different values of α and γ, obtained after 20 learning trials.

Although for simple problems a medium learning rate can yield good results, in general a lower learning rate produces better outcomes. As equation (1) shows, if α is big, the current value of $Q(s, a)$ becomes less important, and its next value is mainly determined by the reward and the next state: $r + \gamma \cdot \max_{a' \in A} Q(s', a')$. When the maze is simple, a big discount factor produces better results, but in complex mazes, for small learning rates, a medium value is preferred.

Figure 5 shows the performance change for a small (11x11) single solution maze, when the learning rate is constant (0.1) and the discount factor varies. One can notice that the number of learning episodes is smaller when the discount factor is small, i.e. the agent concentrates on immediate reward. However, the number of total steps to find the solution has an opposite behavior, as shown in paragraph 2.

Figure 6 shows the performance change for the same maze, this time keeping the discount factor constant (0.5) and varying the learning rate. In this case, both the number of episodes and the number of total steps to the solution are minimum when the learning rate is medium.

4 Path-Finding Using Learning Real-Time A* (LRTA*)

A classical approach for determining the minimal cost path is A* [3], an off-line search algorithm which computes an entire solution path before executing the first step in the path. However, in a multiagent system, it is not often possible to perform local computations for all nodes, due to time limitations. The LRTA* algorithm [5] addresses this problem

Table 2. The performance of Q-learning with state marks for different learning rates and discount factors.

Maze type	α	γ	$m_{episodes}$	$\sigma_{episodes}$	$m_{totalsteps}$	$\sigma_{totalsteps}$
small (11x11) single solution	0.1	0.1	7.6	2.458	14,780	6,048
		0.5	8.15	2.574	8,374	2,797
		0.9	12.8	4.082	9,368	2,284
	0.5	0.1	5.5	2.674	6,272	2,734
		0.5	5.7	2.61	4,124	1,230
		0.9	6.95	1.658	4,292	724
	0.9	0.1	13.55	13.204	11,861	12,152
		0.5	21.8	17.882	15,513	12,753
		0.9	6.05	1.687	4,042	1,139
big (25x25) single solution	0.1	0.1	20.95	7.074	369,139	147,502
		0.5	14.95	4.189	131,756	50,757
		0.9	47.15	16.135	183,783	32,608
	0.5	0.1	>1000		~5,500,000	
		0.5	>1000		~8,300,000	
		0.9	>1000		~4,500,000	
	0.9	0.1	>1000		~18,300,000	
		0.5	>1000		~12,800,000	
		0.9	>1000		~8,100,000	
big (19x30) multiple solutions	0.1	0.1	71.7	19.662	78,764	19,135
		0.5	70.4	32.469	73,511	26,015
		0.9	78.15	19.764	78,814	13,126
	0.5	0.1	>1000		~1,400,000	
		0.5	>1000		~1,200,000	
		0.9	46.8	9.026	46,736	6,517
	0.9	0.1	>1000		~1,600,000	
		0.5	>1000		~1,400,000	
		0.9	391.7	385.956	319,561	306,239

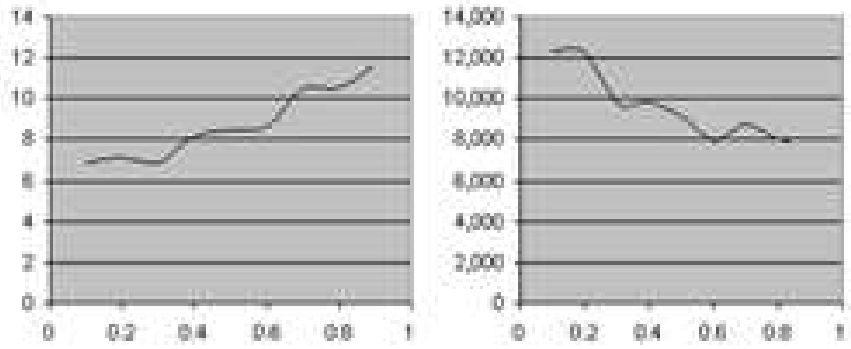

Figure 5. a) The number of episodes when $\alpha = 0.1$ and γ varies; b) The number of total steps when $\alpha = 0.1$ and γ varies.

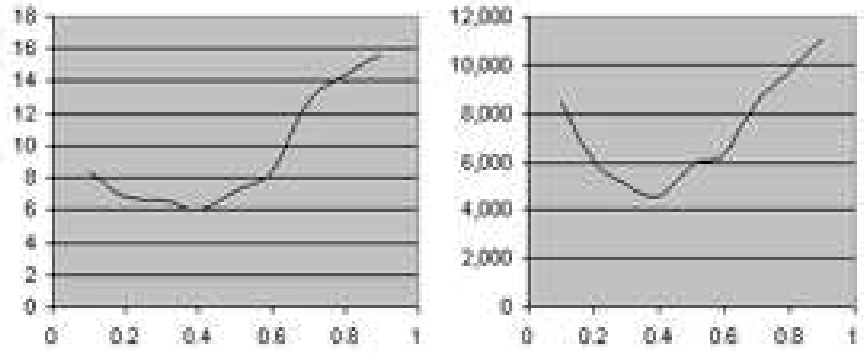

Figure 6. a) The number of episodes when $\gamma = 0.5$ and α varies; b) The number of total steps when $\gamma = 0.5$ and α varies.

by imposing that an agent should perform computations only for its neighboring nodes. Using multiple iterations, it has been proven that the solution eventually converges to the optimal one. The agent must estimate the distance to the goal $h(i)$ for each node i it visits. The LRTA* algorithm is presented bellow [8]:

`Look-ahead: calculate` $f(j) = k(i,j) + h(j)$ `for each neighbor` j `of the current node` i`, where` $h(j)$ `is the current estimate of the shortest distance from` j `to the goal node, and` $k(i,j)$ `is the link cost from` i `to` j`;`
`Update the estimate of node` i`:` $h(i) = \min_j f(j)$`;`
`Action selection: move to the neighbor` j `that has the minimum` $f(j)$ `value. Ties are broken randomly.`

Because the agent determines the next action in a constant time, LRTA* is considered a real-time, on-line algorithm. It updates the estimations with the best so far for not overestimating the actual cost. Therefore, the initial value of $h(i)$ must be optimistic i.e. never overestimate the true value. It was demonstrated [2] that real-time search methods are powerful sub-optimal search methods that can outperform off-line search methods in terms of total running time.

We used LRTA* to find the solutions for our proposed mazes. We measured the number of episodes needed to reach the goal node, the number of learning steps per episode and the number of total steps in which an agent can find the solution. The results are displayed in table 3.

It is evident that LRTA* has better performance than Q-learning for these path-finding problems. LRTA* is an algorithm specially designed for this type of problems, whereas reinforcement learning methods are general learning solutions.

5 Learning Using State Mark Gradients

If the agent marks the states it has visited, by analogy with the real world, the marks should depreciate and this could be used by the agent

Table 3. The performance of LRTA* for the three proposed mazes.

Maze type	$m_{episodes}$	$\sigma_{episodes}$	$m_{steps/ep}$	$\sigma_{steps/ep}$	$m_{totalsteps}$	$\sigma_{totalsteps}$
small (11x11) single solution	15.3	0.461	47.111	23.428	720.8	7.369
big (25x25) single solution	111.2	0.402	213.439	87.273	23734.4	45.026
big (19x30) multiple solutions	96.6	23.02	77.172	81.232	7454.8	951.227

in finding its path to the solution. The agent still goes through an exploration and an exploitation phase. The main idea is that in the first phase, it constantly tries to go to a minimum mark state, which correspond to an unvisited state. In the latter, it tries to follow the ascending state marks gradient, which leads it from the initial state to the goal state.

We propose the following algorithm:

$episode = 1$;
initialize state marks with 0;
initialize state s with the initial state;

while (s is not terminal)
  // choose action
  get state marks for the neighbors: $north, east, south, west$, ignore the walls;
  $wantedDirection = \min(north, east, south, west)$;
  choose the action corresponding to the $wantedDirection$;
  if (more actions possible)
    favor the previous direction with a probability given by $consistencyRate$;

```
      other ties are broken randomly;
   end if

//perform action
   get next state s';

// update state marks
   for all states t
      marks(t) = marks(t)/decayRate;
   end for

marks(s) = 1;
   s = s';
   if (s = initial state and for all neighbors
   sn, marks(sn) > 0)
      episode = episode + 1;
      restart
   end if
end while
```

As stated before, the agent must choose an action that will place it in an unvisited state, or at least a state visited long before. After finding the solution, the agent has the trace of its path, and therefore it must only follow it from the initial state, using the gradient ascent of the state marks. Due to the depreciation, newly visited states have greater marks. The last visited state is the solution.

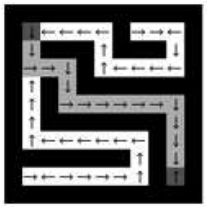

Figure 7. The map of the solved simple single solution maze.

The last test of the algorithm prevents the cases in which the agent

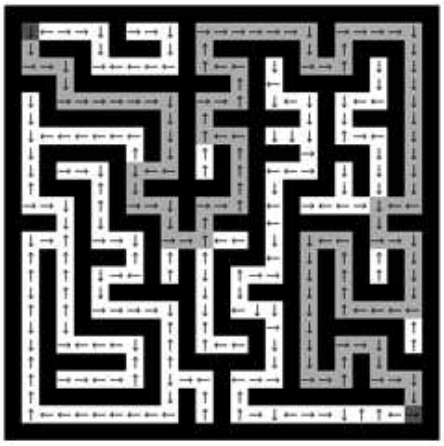

Figure 8. The map of the solved complex single solution maze.

falls into a loop. This situation is not possible for single solution mazes (figures 7 and 8), but it is inevitable for multiple solution mazes (figure 9). If, after a loop, the agent comes back to the initial state, the state marks are reset and a new learning episode begins.

The exploitation phase is done using the same algorithm, but in this case the action at a certain moment is chosen by considering the *max* function instead of *min*:

$$wantedDirection = \max(north, east, south, west);$$

Two constants are used: the decay rate and the consistency rate. The decay rate shows how quickly state marks degrade. The value we used is 1.01. Its purpose become clearer if the agent ignores the marks states below a certain limit. The second constant represents the probability that the agent will keep its current direction, if it is possible. Table 4 shows the results of learning with different consistency rates, after 100 learning trials.

It is obvious that in the case of a single solution maze, where the path is very strict, the consistency rate is not important. The agent will be constrained by the maze design and the only decisions it can make

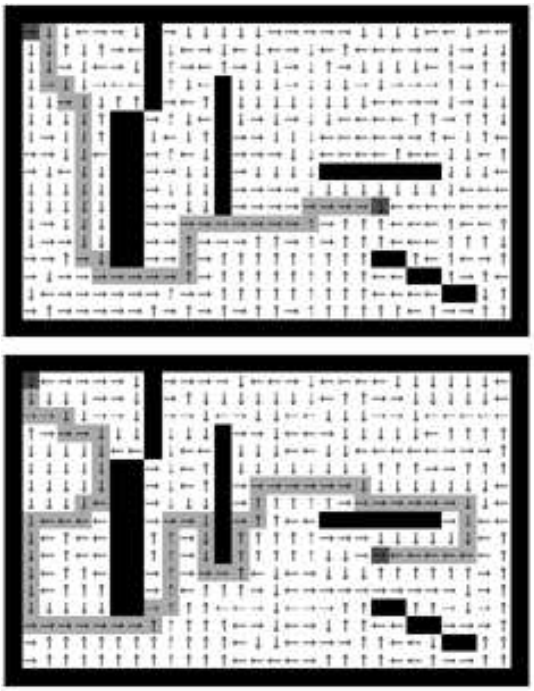

Figure 9. The map of the solved big multiple solution maze: two possible solutions.

Table 4. The performance of state mark gradients algorithm with degrading state marks for different consistency rates.

Maze type	**Consistency rate**	$m_{episodes}$	$\sigma_{episodes}$	$m_{totalsteps}$	$\sigma_{totalsteps}$
small (11x11) single solution	0.9	1	0	68.8	13.717
	0.8	1	0	58.8	22.454
	0.5	1	0	50.6	21.514
	0.1	1	0	50.32	23.981
big (25x25) single solution	0.9	1	0	260.4	80.039
	0.8	1	0	261.6	87.447
	0.5	1	0	237.2	76.578
	0.1	1	0	242.48	86.238
big (19x30) multiple solutions	0.9	2.9	2.385	614.56	505.517
	0.8	3.25	2.875	612.76	462.663
	0.5	4.25	3.708	637.92	450.715
	0.1	6.36	5.572	811.28	544.777

refer to cross points. For big multiple solution mazes, if the consistency rate is high, the number of total steps to reach the solution is lower, because the agent will travel less in a random manner, thus increasing its chances to find its goal more quickly.

Our algorithm proves to be more efficient than Q-learning and LRTA*, as it discovers the solution in much fewer learning episodes (only one episode if there are no loops possible in the maze). Also, the number of steps in which the agent reaches the goal node is up to ten times smaller than the number of steps necessary for the LRTA* algorithm.

6 Conclusions

As shown, the proposed method has better performance than the other presented algorithms. This is validated both by the theoretical approach as the practical results that are depicted in tables 1 to 3. The main advantage of this method is that in a multiagent system the other agents can use the results about discovered paths by simply inspecting the current location marking. The method can be easily modified in

order to be used over generalized graphs.

References

[1] Bhanu B., Leang P., Cowden C., Lin Y., Patterson M., *Real Time Robot Learning,* Center for Research in Intelligent Systems, University of California, http://www.mark.vcn.com/r_learn.html

[2] Cakir A., Polat F., *Coordination of intelligent agents in real-time search,* Expert Systems, Vol. 19, No.2, 2002

[3] Hart P. E., Nilsson N. J., Raphael B., *A formal basis for the heuristic determination of minimum cost paths,* IEEE Transactions on System Science and Cybernetics, 4, pp.100–107, 1968

[4] Kaelbling L. P., Littman M. L., *Reinforcement Learning: A Survey*, Computer Science Department, Brown University, Providence, http://www-2.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/rl-survey.html

[5] Korf R. E., *Real-time heuristic search,* Artificial Intelligence, 42, pp.189–211, 1990

[6] Sutton, R. S., Barto, A. G., *Introduction to Reinforcement Learning,* MIT Press/Bradford Books, Cambridge, Massachusetts, 1998

[7] Watkins, C. J. C. H., *Learning from Delayed Rewards,* PhD Thesis, King's College, Cambridge University, 1989

[8] Yokoo M., Ishida T., *Search Algorithms for Agents in G. Weiß (ed.): Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence,* The MIT Press, Cambridge, Massachusetts, 2000

Florin Leon, Mihai Horia Zaharia, Dan Gâlea, Received February 1, 2004

Department of Automatic Control and Computer Engineering
Technical University "Gh. Asachi"
Iaşi