

Implementation and testing of Markov Decision Process planning algorithm

Jaromír Dvořák

Abstract— MDP agent planning method for non-deterministic environments. Java implementation of value iteration algorithm and testing with various parameters. Time and computational efficiency statistics.

I. INTRODUCTION

THIS report was written as a semestral work for subject A3M33UI, CTU FEE Prague.

In agent technology, one of the main issues is planning. In a discrete environment, an agent is always in certain discrete state. It is needed to choose from a set of actions, which to perform next, to reach the goal state. The **Markov Decision Process** algorithm is used in cases of planning, where transition between two states is not deterministic process. The value-iteration implementation of MDP creates utility function, i.e. a number is assigned to each state. Agent then chooses to move to a state with highest utility number.

We will use a simple grid world here as our environment. Each field represents one state. There are only 4 possible actions: go north, go west, go east, go south. The probability of correct transition was set to 0.8. Remaining divided among the neighbour states according to the experimental platform.

II. METHODOLOGY

A. Setup of experiments

At first, the algorithm was implemented in Java according to the abstract definition in [1] using the interfaces described in [2]. Penalty parameter was added in the algorithm to the reward of all states, except of dummy states, in which the absorbing states leads to. A simple maze (labyrinth) was created using the experimental platform's maze editor, as our environment. Several tests were then performed and the results are presented below. At the end, time computational complexity was measured and plotted for some of the parameters configuration, according to the submission.

B. Configuration of algorithm

The algorithm was tested on the maze with given various parameters (gamma, penalty, epsilon), chosen from ranges:

- Discount factor (gamma): 0.5, 0.8, 0.9, 0.99, 1
- Penalty (cost of each step) -1.0, -0.3, 0, 0.5
- Maximum error (epsilon): 0.2, 0.1, 0.05, 0.01

The following maze was created for testing the algorithm. The maze has one absorbing state, one with negative reward and one teleport state. Size of the rewards were chosen in respect to the maze width and height, as well as to the range of given discount and penalty values, to show reasonable results.

Set of used software tools:

1. **MDP Testbed v0.3**
2. **Eclipse IDE**
3. **Java SE JDK**
4. **Gnuplot**

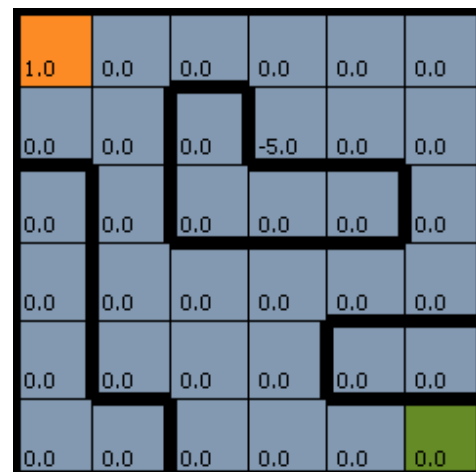


Fig.1 The experimental maze

III. EXPERIMENTS

The computed utility and strategy functions

The epsilon parameter was fixed to 0.01, because of it generally affects only the accuracy, and plot the utility and strategy functions for several combination of parameters.

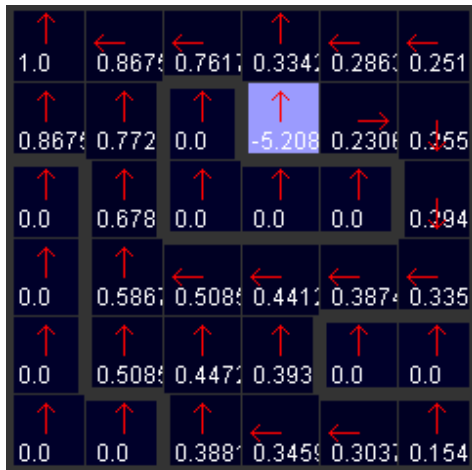


Fig.2 Gamma = 0.9, Penalty=0



Fig.5 Gamma = 0.99, Penalty = -0.3

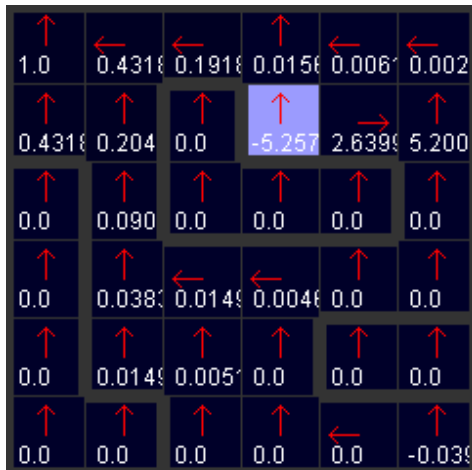


Fig.3 Gamma = 0.5, Penalty = 0

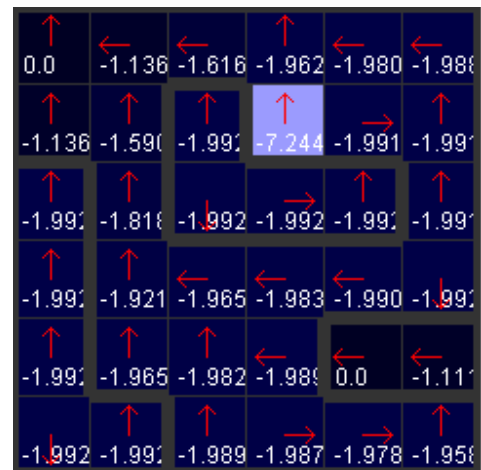


Fig.6 Gamma = 0.5, Penalty = -1

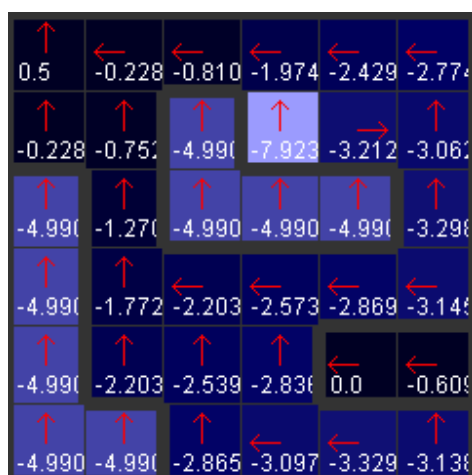


Fig.4 Gamma = 0.9, Penalty = -0.5



Fig.7 Gamma = 0.9, Penalty = 0.5

First plot (fig.2) shows a utility function for environment, where movement is free of cost. The state with negative reward is avoided by the algorithm, even from the states situated closely to the goal.

In next plot (fig.3), the discount factor was decreased to 0.5, which could suggest an environment that can vary over a time, or planning for far future is not affordable. Here we can see that the algorithm “doesn’t see” the way around wall object and leads to go towards the goal near the negative rewarded state and undergoes risk to fall into it. Note that in the closest state, the policy leads to go up instead of to the left, because of the probability to fall to the negative rewarded state is smallest by choosing this action.

Third plot (fig.4) shows an environment, where movement is not free of cost, but each step takes -0.5 of reward. Same as before, the algorithm leads to go near the state, moreover in the closest state it decides to move left directly.

Following plot (fig.5) shows an utility function with really high discount: 0.99. This means planning for far future and would be suitable really “static” environment. Computational load increases with gamma factor. Penalty was set to -0.3 for each step. We can see only minor changes in the policy behavior, comparing to previous two plots.

Fifth plot (fig.6) displays environment, where movement is even more expensive and discount factor is as low as 0.5, so far states are less taken into account. Here it is well shown that for states close to the teleport, policy leads towards it, because of it is far from the goal. By entering the teleport state, the probability of appearing in more close state to the goal is then high.

The last plot (fig.7) shows a paradox environment, where each move brings up positive reward. Here we can see, that agent behaves repulsive to the absorbing state, because of it can earn only by moving. The actual utility of states increases exponentially with discount factor.

Computational complexity of the designed algorithm

The algorithm was additionally tested for computational complexity, and dependency of number of iterations to the epsilon and gamma parameters.

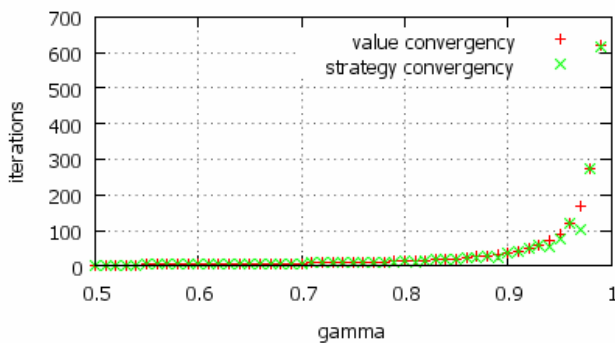


Fig.8 – convergency dependence on Gamma parameter, epsilon = 0.1

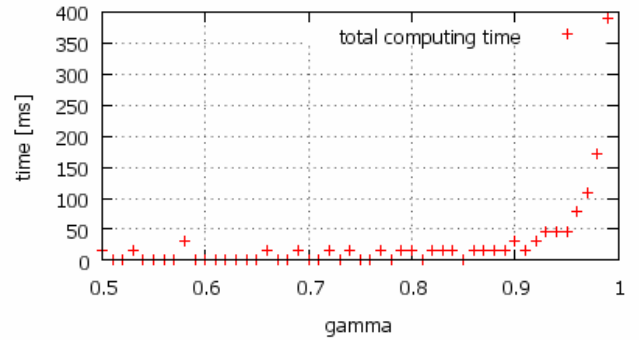


Fig.9 – total computing time dependence on Gamma parameter, epsilon = 0.1

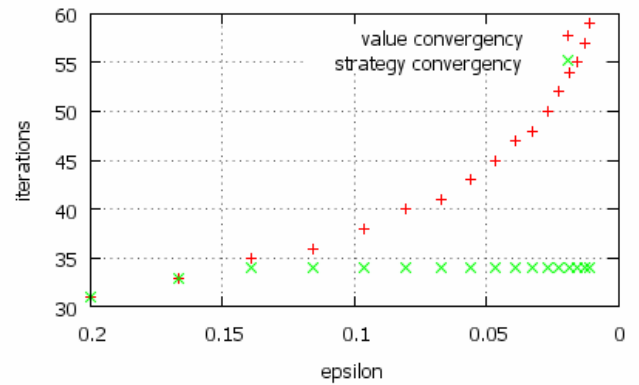


Fig.10 – convergency dependence on Epsilon parameter, gamma = 0.9

Gamma parameter (discount factor) denotes how far the planning algorithm “sees”. The actual reward of another state is taken into account proportionally to $\exp(\text{distance}/\text{gamma})$. Computational complexity arises with this factor. First plot shows the convergency dependence on gamma parameter. Policy usually converges in fewer steps than utility value, ie. since certain iteration, it remains constant. However, we’re using here some states surrounded by walls, from which the agent cannot reach the goal state. These states are causing oscillations in policy during computation, that’s why the strategy convergency arises together with value convergency, in the first graph.

The value convergency (Fig. 10) depends on epsilon parameter also exponentially, as expected. The epsilon parameter gives us the maximum delta, in which must the utility for each state not differ in each iteration step. Thus, this parameter denotes precision of the computing. Here it is clearly shown, that policy (strategy) converges in fewer steps than utility value, ie. since certain iteration, it remains constant. This is the minimum value for epsilon for the algorithm to work correctly.

IV. CONCLUSION

This work is showing the purpose, pitfalls and basic tuning of MDP agent planning technique. The MDP value iteration algorithm was implemented in Java. Simple maze (labyrinth) was created, as our environment. Several tests were then performed on the algorithm. In the end, time computational complexity was measured and plotted for some of the parameters.

REFERENCES

- [1] Michal Jakob, Lectures for A3M33UI :
(https://cw.felk.cvut.cz/lib/exe/fetch.php///courses/a3m33ui/prednasky/files/ui-2010-p10-markov_decision_processes.pdf),” in *Plastics*, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.
- [2] Petr Benda, Notes for algorithm implementation:
https://cw.felk.cvut.cz/doku.php/courses/a3m33ui/ulohy/podrobne_implementationaci_pokyny.
- [3] Semestral work submission:
<https://cw.felk.cvut.cz/doku.php/courses/a3m33ui/ulohy/>