CS7461 Machine Learning: Assignment 4

Matthias Grundmann
grundman@cs.tum.edu

1 Introduction

In this assignment I analyze two different Markov Decision Processes (MDP). The first problem has few states and is easy to solve, and the second one has thousands of states. Value iteration and policy iteration is used to obtain the results. These function are provided by the MATLAB MDP toolbox from INRA. [1]

1.1 Used MDPs

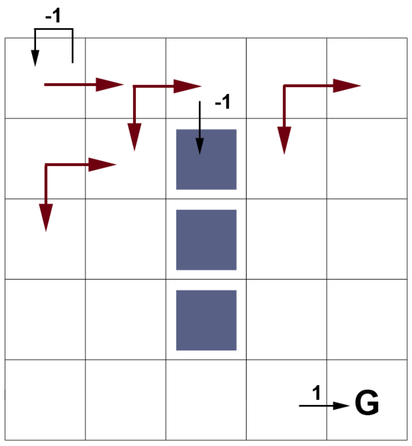
I used a variation of the grid-world example given in the lecture and the car racing example first described in [2].

1.2 Yet another grid-world

The beauty of the simple grid-world example is, that it is easy to check, whether the solution obtained by the two algorithms is optimal and the number of states is quite limited. It also can be easily extend in its size to determine how well both algorithms scale with respect to the problem size. So it is a good starting point for the comparison of both algorithms.
Furthermore the grid-world can be easily extend to capture real world problems. For example the environment of a robot that is moving in a plane can be modeled as a grid-world. Different goals and tasks can be described by appropriate reward functions. Other applications include scheduling tasks, e.g. using a taxi fleet optimally to transport passengers or using robots in a warehouse to transport goods. Figure 1 shows the two grid-worlds I used for this assignment. The agents starts anywhere at a white, i.e. unblocked, tile and tries to reach the goal in the bottom right of each grid-world. It can execute 4 different actions at each time step: Left, Right, Up and Down. The agent does not get a any reward when moving from a free tile to another free tile, but it gets a reward of 1 when moving from a free tile, that is adjacent to the goal tile, to the goal. When the agent tries to move outside the world or into an obstacle it stays in the same state but gets a reward of -1. So the agent will try to avoid hitting any obstacles and will try to find the shortest way to goal. To make the setting more realistic and robust to errors, the agent executes with a probability p the action it intends to execute, and will move in another direction with probability $1-p$. The discount rate is set to 0.9. Please note that at some tiles the agent has different choices, that are equally optimal. Examples are represented by the red arrows in figure 1. It is interesting to investigate which paths for the optimal policy will be computed with respect to different choices of p. I used two different grid-worlds to examine how the iterations of each algorithm change, when the world is more constrained, for example the path in the right figure 1 should be faster computed, because only two of four actions are valid.

1.3 Car race

This is example is first described in [2] and is already implemented in [1]. The problem is that we are given a course with a start and finish line and asked for a optimal sequence of acceleration and brake events to accomplish the course as fast as possible. Obviously this example has many real world applications, e.g. in robot navigation, car race analysis or optimal planning strategies. It is also very interesting because it models a human driver on a circuit that is not perfect, i.e. with a probability p the intended action will not be executed. Because a human driver has a specific response time to



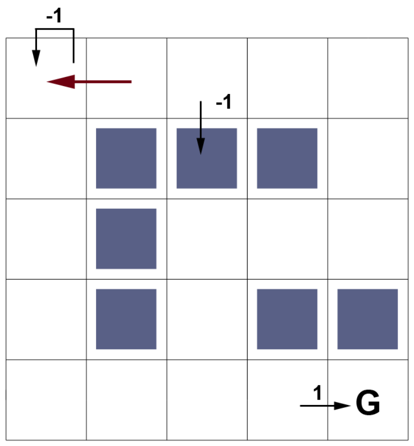


Figure 1: The 2 used gridworlds. Left: simple, right: intermediate

recognize events and act accordingly, he may fail to act correct in a specific moment.
Furthermore we will see, that even when we represent the circuit by a small and coarse grid, the state space will be so huge that some algorithms can not be applied. Figure 2 shows the tracks that I used for this assignment. The gray pixels represent the road, the white pixels represent the start/finish line and black pixels represent the border. A driver always starts at the start/finish line and has to reach the finish line, which is an absorbing state. The car state is modeled by a combination of its position and its current velocity. More specifically its state is represented by a 4-tuple $(u_x, u_y, \dot{u_x}, \dot{u_y})$, where (u_x, u_y) represents its current position and $(\dot{u_x}, \dot{u_y})$ its velocity in x and y direction. All values are constrained to be integers.
At each time step the car can accelerate, brake or keep its velocity constant. To simplify the problem the car is only allowed to change the velocity by 1 unit in each direction independently, i.e. the acceleration must be in {-1,0,1}. This leads to a total of 9 different possible actions. The question arises how big the state space is? For example a car on the left track has $10 \cdot 10 = 100$ possible positions, but only 55 of these are inside the border. We limit the velocity to a maximum magnitude of $\dot{u_m ax}$=3. That means that the highest possible discrete velocities are of the form $(\pm 3, 0)$ or $(\pm 2, \pm 2)$. We have 4 directions, so 4 velocities of the form $(3, 0)$ and in the range of $-2..2$ are 5 integers, leading to 25 possible velocities, in total we have 29 discrete velocities with a magnitude less or equal than $\dot{u_m ax}$. So the state space has $29 \cdot 59 = 1595$ different states. So the transition matrix has $1595 \cdot 1595 \cdot 9 = 22.89$ million entries, the reward matrix $\cdot 1595 \cdot 9 = 14355$ entries. Although these matrices are sparse, we can conclude that the policy iteration is hard to apply to this problem, because solving linear equations with more than 10.000 unknowns is very memory intensive and takes a lot of time. It was not possible to apply policy iteration to the right race track with $319 \cdot 9 = 9251$ states, cause the linear system are too big.
As noted above the problem also models human response time by including a 10% failure probability for each action.
To complete the track as fast as possible we want to compute a solution with the fewest time steps, the distance is not important. That is why at each time step the driver receives a ”penalty reward” of -1. If the driver hits the border or uses a shortcut crossing the border he receives a penalty reward of -100. The discount rate is set to 0.99, cause we only focus on how many time steps are needed to reach the finish line. Maximizing the reward should lead to the fastest and optimal track time.

2 Results

2.1 Yet another grid-world

Both algorithms compute the same optimal policy for the easy and the intermediate grid. The optimal policy for both grids is shown in figure 3. Obviously the calculated optimal policy is a correct optimal policy. Table 1 shows the run-time and the iterations each algorithm need to compute the

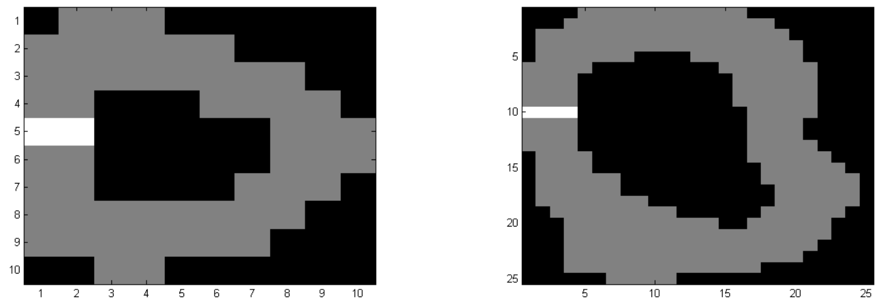


Figure 2: The 2 used race tracks. Left: 10x10, Right: 25x25

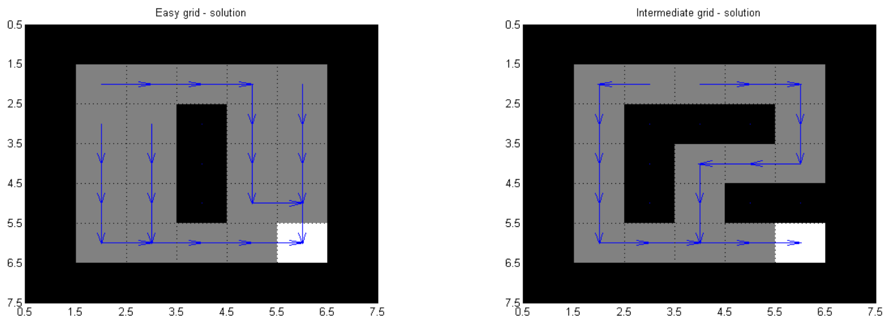


Figure 3: The optimal solutions for the grid-world. Left: simple, Right: intermediate

optimal policy. Value iteration needs around 5-10 times more iterations and 3-4 times execution time. Because policy iteration solves a linear system and does not refine the policy in a greedy manner, it uses more time per iteration step, that is why there is a discrepancy between the factors of iterations and execution time.

Algorithm	Problem	Iterations	Time
Value	Simple grid	17	0.04
Policy	Simple grid	3	0.01
Value	Interm. grid	21	0.01
Policy	Interm. grid	2	0.03

Table 1: Run-time and iterations for the grid world

Figure 4 shows the v-values, that is the expected reward of each state following the optimal policy. It can be seen, that the closer the state to the goal the higher the v-value. This makes totally sense, because the faster the agent reaches the goal, the higher its reward should be. It is interesting that the v-value of tiles that are adjacent to the goal is 1.09 instead of 1. This is because it is the expected reward following the optimal policy and there is a 10 percent chance not to execute the action the agent want, leading to other ways to the goal. Setting the error probability to 0 leads to an v-value of 1, as expected.

I also examined which effect to optimal policy different choices of the probability p have, i.e. the probability to execute the intended action. Figure 5 shows on the left the result for $p = 1$ and on the right the result for $p = 0.6$. $p = 1$ shifts the solution at the top one tile to right, but is still optimal, the distance of the top tile to the goal is same whether the agent goes first right or left. Interesting is the choice of $p = 0.6$, that means the probability to execute the intended actions is just slightly above average. That is why the solution shifted on tile to the left compared to $p = 0.9$. Now it is

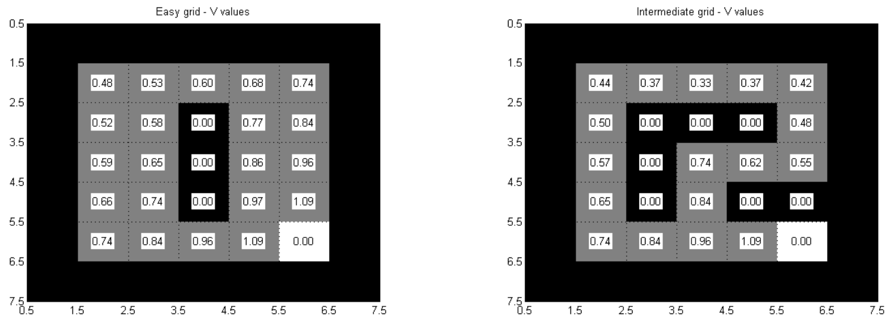


Figure 4: V-values for the grid-world. Left: simple, Right: intermediate

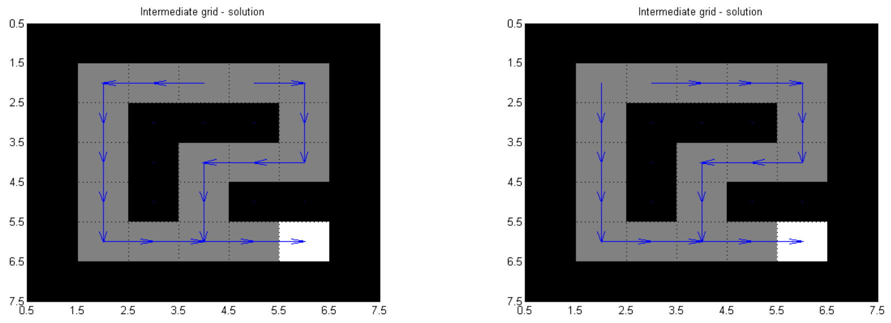


Figure 5: Solution for different choices of p Left: $p = 1.0$, Right: $p = 0.6$

saver for the agent to move to the right starting from the 2nd tile in the first row, because also left seems to lead to a path that is shorter, if the move after left does not follow the optimal policy the agent has to go a long detour. Because both algorithms select the starting position to calculate the optimal policy randomly, each algorithm should be run several times. However the problem seems to be so simple, that the results never change and also converge to a optimal solution. Also the iterations have been found be constant for the problem.

How well does the algorithms scale with respect to the grid size? I extended the grid to a size of 10×10 and 20×20 and run both algorithms again. An example for the optimal policy as well as the v-values (white mean high, black low) for the 20×20 grid is given in figure 6.

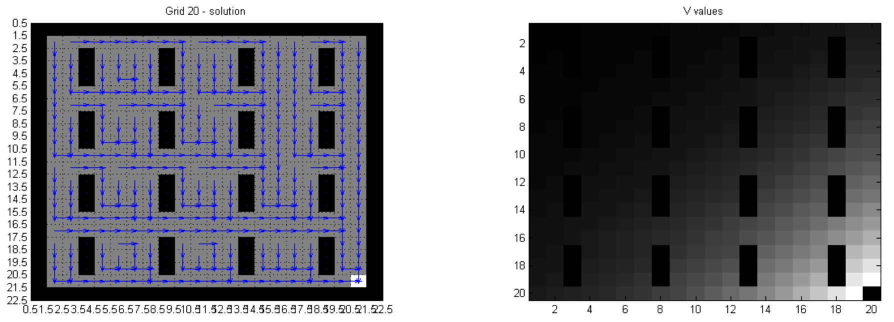


Figure 6: Left: Solution for 20x20, Right: V-values for 20x20

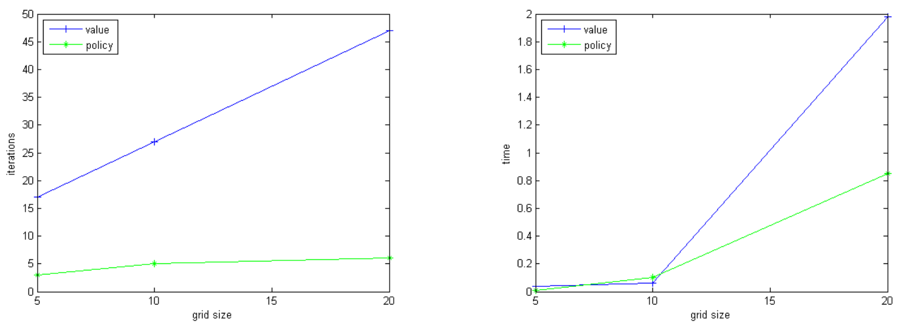


Figure 7: Resources for different grid sizes. Left: Iterations, Right: Execution time

As can be seen in the left figure 7 both algorithm's iterations scale linear w.r.t to the square of the input size, although the factor of value-iteration algorithm is higher than the one of the policy-iteration algorithm. I assume that this because the policy algorithm solves for V-values or respectively the Q-values simultaneously keeping the optimal policy fixed, instead of the solving for them in a iteratively greedy manner like the value-iteration algorithm does. This should lead to fewer iterations because of increased stability, but also each iteration takes more time. This can be seen in the right figure 7. For a grid size of $k = 10$ value iteration actually performs faster than policy iteration, also it uses more iterations. For higher values the instability of the value-iteration approach seems to outweigh its faster iterations. The figure also shows that the algorithms scale quadratically w.r.t. the square of the input size, i.e. they scale linear in time with w.r.t. to the input size.

2.2 Car race

The Car race problem shows that policy iteration is not always the preferred algorithm. If the state space is huge and we have a small number of actions compared to that, the transition matrix will be very sparse. Solving the linear system for the Q or respectively V values can take unnecessary long. Also as noted in the introduction the matrix may become so big that it is impossible to solve the system directly[1]. Table 2 gives the result for different runs of both algorithms. First one noticed that unlike the grid-world examples, the results are different for each run. This is because the start points to calculate the optimal policy are chosen randomly by the algorithm. Second value iteration is around 2 times faster than policy iteration, although it needs again more iterations. For the race-problem of grid size 25×25 policy iteration could not be applied, but value iteration calculated a route in 157.85 s using 36 iterations.

Algorithm	Time steps	Execution Time	Iterations
Value	15	1.47	27
Value	15	2.62	27
Value	14	3.8	27
Policy	15	5.68	8
Policy	16	5.65	8
Policy	14	5.64	8

Table 2: Run-time and iterations for car race problem for 10×10 grid

Different examples for the optimal race tracks for policy iteration are given in figure 8 and for value iteration in figure 9. Green arrows show the velocity and red arrows the current acceleration.

Figure 10 show the race track obtained by value iteration for the 25×25 problem

[1]although iterative solvers could be used, but this approach was not followed in this assignment

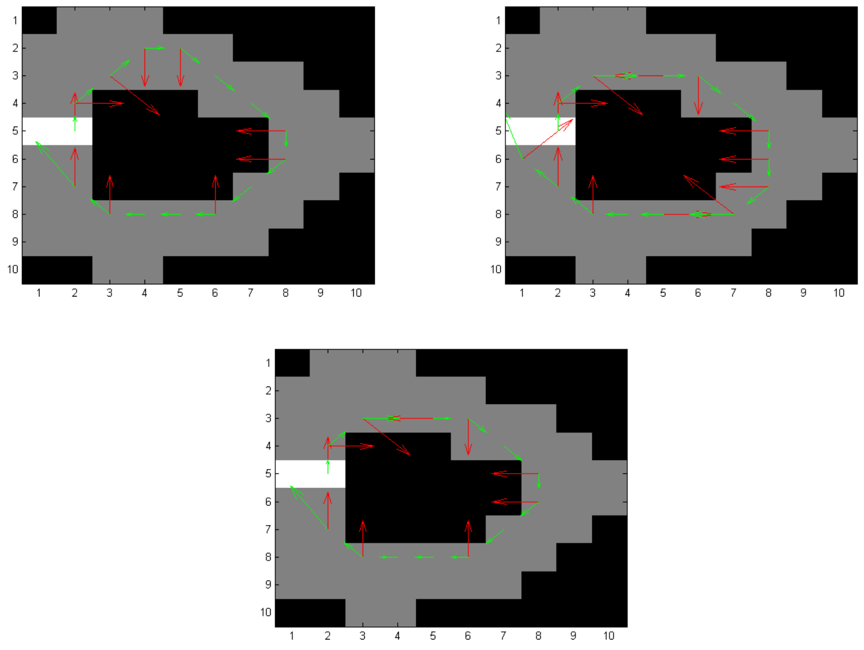


Figure 8: Examples for different race tracks of length 15,15 and 14 for value iteration

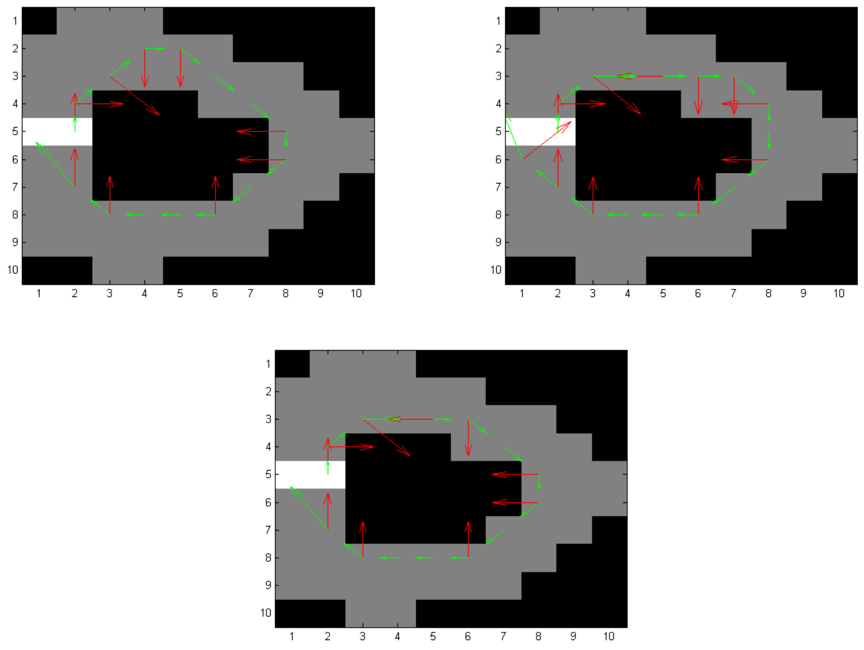


Figure 9: Examples for different race tracks of length 15,16 and 14 for policy iteration

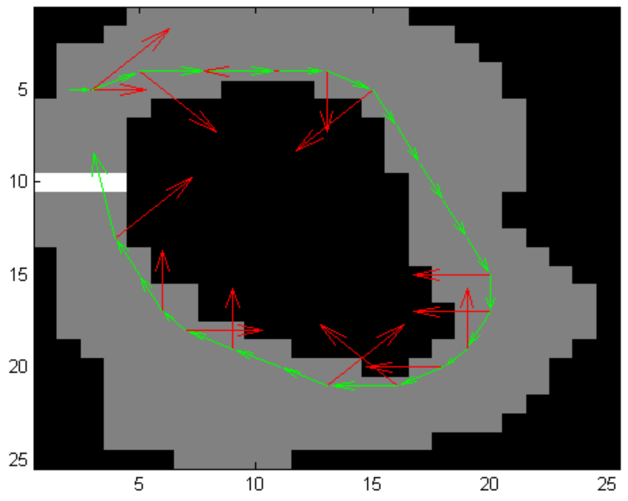


Figure 10: Examples for race track of size 25 $\times$ 25 for value iteration

3 Conclusion

It could be shown that on small size problems in general policy iteration outperforms value iteration cause of its increased stability. When considering larger problems, the resources policy iteration needs to solve for the optimal policy become disadvantageous and value iteration can perform faster. I also showed that sometimes the problem size is prohibitive huge to apply policy iteration.

Regarding MDPs we can summarize that a careful design of the transition and reward matrix can lead to optimal policies by both algorithms. However, during this assignment I found that both algorithms are very sensitive to small errors or wrong parameters.

References

[1] http://www.inra.fr/internet/Departements/MIA/T//MDPtoolbox/.

[2] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artif. Intell.*, 72(1-2):81–138, 1995.