Markov Decision Processes

Corentin Godeau

11/26/2018

1 Introduction

In this assignment we will study Markov Decision Processes by applying Reinforcement Learning on them. For this assignment I will use the java library BURLAP[1] developed at the Brown university. For the second part, I used a python library, mdptoolbox[2], to solve the mdp using Q Learning.

My first work for this assignment was to find two Markov Decision Processes with the constraints of one having a "small" number of states and the other one significantly more states. What I come up with is a very well known problem : The Frozen Lake. Assuming a grid based world, each cell of this grid can have one of the following types:

- Start cell: The starting point
- Goal cell: The destination where we want to go
- Frozen cell: it's safe to visit this cell
- Hole cell: it's not safe to visit this cell

An agent walking on the lake wants to go from the start cell to the goal cell without falling into a hole. Furthermore, as the lake is frozen, the displacements are not precise and there is a probability to reach the wrong cell.

This problem is easily modelized by a Markov Decision Process. The cells represent the states, the actions are the direction are the actions that an agent can take, the rewards are defined by the type of cell the agent walks on and there is a probability distribution over the state reached.

What's interesting about this problem is that it's quite easy to see a real application in real life, it has a practical aspect. It's also a very known problem that has been studied for a long time. Furthermore, it has no definite size. We can scale the world grid up to a very large number, giving really a lot of states. The same problem is coherent with the constraints we had for this assignment.

2 Solving Markov Decision Processes with VI and PI

The first problem I decided to study was the previously described Frozen Lake problem with a grid size of 4 by 4. It has the following layout:

Frozen	Frozen	Frozen	Goal
Frozen	Hole	Frozen	Hole
Frozen	Hole	Frozen	Hole
Start	Frozen	Frozen	Frozen

As we can see, there are two possible paths reaching the goal cell and a "trap" in the bottom right corner. In term of reward, every frozen cell has a reward of -0.04, the holes have a value of -1.0 and the goal cell has a value of 1.0. Finally I added a bit of randomness by having a probability of 10% to go in the wrong direction.

The first experiment I did was to compare the Value Iteration and Policy Iteration algorithms on this particular problem. I computed the policy map giving the action to take at each state. For the Value Iteration, the algorithm came up with this layout in 9 iterations:

Figure 1: Value Iteration

For the Policy Iteration, the algorithm came up with this layout in 4 iterations:

Figure 2: Policy Iteration

We need to notice that BURLAP seems to use the Value Iteration algorithm to perform the Policy Iteration algorithm. To compare the two algorithms, it seems that the Value Iteration algorithm converges faster in term of iterations as it requires only 9 iterations to converge to the final solution. I did multiple experiments on the compute time by measuring the average time needed to converge. On 1000 repetitions, it seems that the Policy Iteration is really slower with 22 ms per iteration compared to 1.5 ms for the Value Iteration. We know that the Policy Iteration is more expensive in term of compute time for each iterations but often needs less iterations before converging. Here it seems that the fact that it needs less iterations is not sufficient to reduce the compute time efficiently. We'll see if that's also the case with the more complex problem.

The two algorithms do not converge to the same solution, however we see that there is only one difference in the upper right corner and the two best actions give the same result.

The second problem I decided to study is a bigger version of the previous one. It has a grid of 24 by 24 (576 states) and the following image describe the layout:

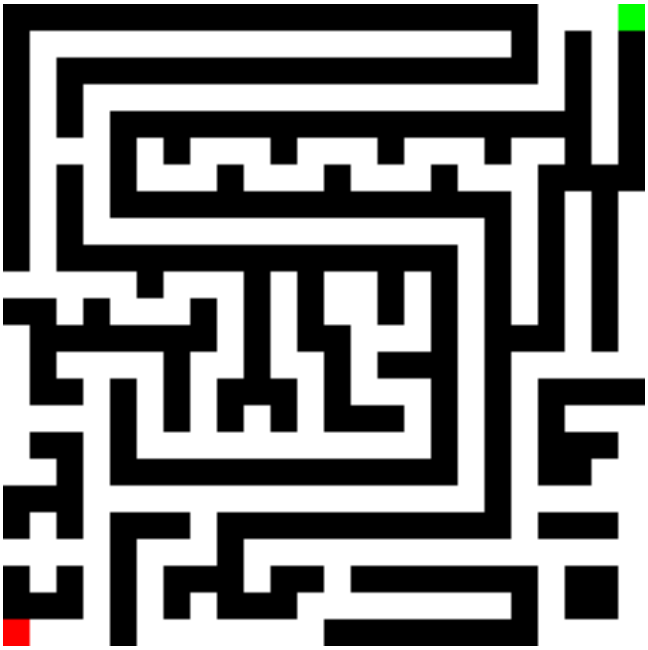

Figure 3: Large problem map

The white cell represents the frozen ones, the black cells the holes, the red cell is the starting cell and the green cell the goal. I designed this problem like a maze, bu we need to notice that it doesn't really act like one because, you can walk on a black cell, it just gives a stronger negative reward for doing so.

I did exactly the same experiments and here are the resulting solutions:

Figure 4: Value Iteration

Figure 5: Policy Iteration

The two layouts seem to be approximately identical. It's quite hard to read, but we can notice an interesting artifact. I designed the problem to be like a maze, I also explained that it doesn't really act like a maze because one can walk on a "wall" cell. This is why we observe the fact that sometimes the best action is to cross a hole. If we would change the reward of the hole to be really negative. Doing this would mean that a long path of frozen cell is better for the total reward. In term of iterations, the Value Iteration need more iterations but as the Policy Algorithm uses the Value Iteration, the total number of "basic" iterations for the Policy Iteration algorithm is 1124 which is clearly higher. I don't really know how to explain that, I thought the difference would be visible on a larger problem. My guess is that this particular problem is well adapted for the Value Iteration algorithm. This difference is also clearly visible in term of computing time, the Policy Iteration is about 3 times slower than the Value Iteration.

3 Q-Learning

After having used the Value Iteration and Policy Iteration algorithms, I needed to try a RL algorithm to solve the two problems I designed. I decided to try the Q-Learning algorithm because we saw how it works and it is quite easy to understand the principle.

My first try was on the simple problem with only 16 states. I decided to use a discount factor of 0.9 and did 1000000 iterations. Here is the result I obtained:

Figure 6: Q Learning

Globally we can see that the policy is quite similar to the previous ones we got using the VI and PI algorithms. However, there are minor differences that could lead to a smaller reward due to the probabilistic transitions. In term of iterations count, this method is clearly slower than the two previous ones. I did not manage to get satisfactory results with less iterations.

On the second problem, the following figure represents the policy:

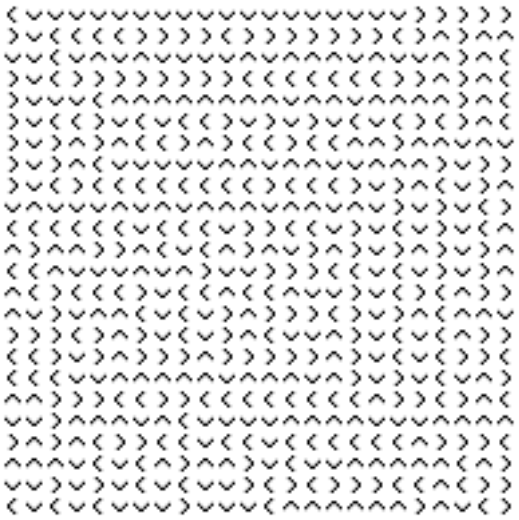

Figure 7: Q Learning

Here we see that the result is really not interesting. We can see that there are patterns appearing, but trying to follow the best directions often don't guide to the goal. Which is problematic because it seems that the learning process didn't manage to understand clearly what was the goal.

In order to improve the results, I tried to tweak the parameters of the algorithm. unfortunately, this was not successful and I did not obtain a better result for the complex problem. Increasing the number of iterations caused the computation to be very long, 10000000 iterations on the large problem took more than 10 minutes, and the result was still not very satisfactory.

It also seems that decreasing the discount factor too much increases the number of iterations needed to reach the same "wanted" policy. However this does not mean that me must set this factor too close to 1. In my case I obtained better results with a discount factor of 0.9 instead of 0.99.

4 Conclusion

In this assignment we saw different approaches to solve Markov Decision Processes and compared them. The first experiments were to compare the classical MDP solving algorithms : Value Iteration and Policy Iteration. The conclusion was that on the two problems I chose, the value iteration seemed to give an interesting policy using less iterations than the other algorithm, thus being the fastest algorithm. However it's interesting to notice that in term of "base" iterations, the PI requires less but use the VI in each passes. That's why the VI is faster, but we may observe another behavior on other problems.

Then, I decided to use a classical RL algorithm, the Q Learning. The core principle is very close to the Value Iteration algorithm. Unfortunately, my experiments were not as successful as the others. On the small problem, the results were correct and similar to the other ones. On the bigger problem however, the algorithm did not produce any interesting results. I don't really know how to explain that but tweaking the parameters did not really change the result and there clearly is a difference between the Q-Learning algorithm and the two other ones. Trying to increase the number of iterations could probably improve the obtained policy but it would really take a long time : 10000000 iterations on the second problem took the algorithm more than 10 minutes to complete.

In conclusion I think it's important to notice that these algorithms are not designed for the same purpose. Here the VI and PI algorithms can find a solution because the rewards and transitions are well defined and we know every variables. It's not always the case, and for these particular situations, the Q Learning and other RL algorithms would probably be a better approach.

References

[1] Burlap. `http://burlap.cs.brown.edu/`.

[2] mdptoolbox. `https://github.com/sawcordwell/pymdptoolbox`.