Project 4 - Markov Decision Processes

Jean-Pierre Bianchi - GT ID jbianchi3

[ ]

1 Choice of MDPs

I have decided to take the small non-grid world Forest Management MDP, MDP1, which comes with mdptoolbox-hiive. Not only is it non-grid, which is a bit harder to process mentally, but the reward matrix has two dimensions since the reward depends on the action we take, ie R(s,a) (which was a bit of a challenge to implement since I coded the algorithms). I wanted a simple problem (as advised in the writeup) to not be distracted by the problem itself (like MDP2) or by complex transition matrices so I could focus on the algorithms themselves. The size is easy to increase so I plan to push it until I see significant differences between the alorithms.

The second MDP, MDP2, is the Frozen Lake environment with size 20x20. I chose it because it has multiple game-ending absorbing states and is highly stochastic. More generally, I think this environment also represents life situations where there is a difficult road to success, paved with many difficulties and traps. The position of the holes will greatly influence the average score and how the algorithm behaves before converging as we will see. For instance too many holes near the start will give poor average score because of the many opportunities for the player to fall into one of them. This is also true if there the optimal policy comes close to many holes, which means the Lake is full of holes, so it's an element I could change to find interesting situation as we'll see later.

I kept the probability of 1/3 from the Gym environment so that the optimal solution is going to have to cleverly avoid the holes by sometimes not going towards the goal, but in a way that all 3 possible resulting states are not holes. But sometimes, the best strategy is to go towards a hole (like in life, at times we have to face tough problems rather than running away from them) because there is a 2/3 probability to not end up in it. For instance, when one is caught between two holes. Obviously the algorithms can deal with these situations, but there are two equivalent best actions here, ie going straight into one or the other, which gives the same resulting value and leads to two different but equivalent optimal policies.

I kept the reward from a hole at zero like in Gym. The goal is absorbing but has a value of 1, so it was not obvious at first how to deal with this because, by looping on itself, the state value was going to skyrocket and was going to change when using different methods or different maps. So, I made it non absorbing, and redirected all the moves from it to a new absorbing state, which it the one that actually wins the game. So, one can pass into the 'Frisbee state' only once, and the resulting utility matrices are nice-looking and comparable.

Since we have to use VI and PI, I needed the transition matrix P and reward matrix R. Another reason why I picked Frozen Lake is that is comes with a function that creates a lake of any size, one that has a solution. This will make my life easier, and also allow me to study the influence of the size of the lake. I generate the P and R matrices by feeding a Lake map into Gym's environment, only to get the env.P dictionary with all the corresponding transitions probabilities, from which P can easily be generated. After that, I do not use the Gym environment at all. I have coded VI, PI and Q-Learning to work with those matrices. Obviously, for Q-Learning, I emulated the step method using the P and R matrices, which takes 2 lines of code. I took this decision so I could use the exact same environment and reward structure as VI and PI, which has the obvious benefit of being able to, say, change the rewards by changing R only, and not worrying about modifying the Frozen Lake code. Nice and simple!

1.1 MDP1

This problem is relatively simple and will give me a first 'training' on the various methods studied, and prepare me for the next, much more complex MDP. Let's look at the convergence and reward curves in Figs 1 and 2.

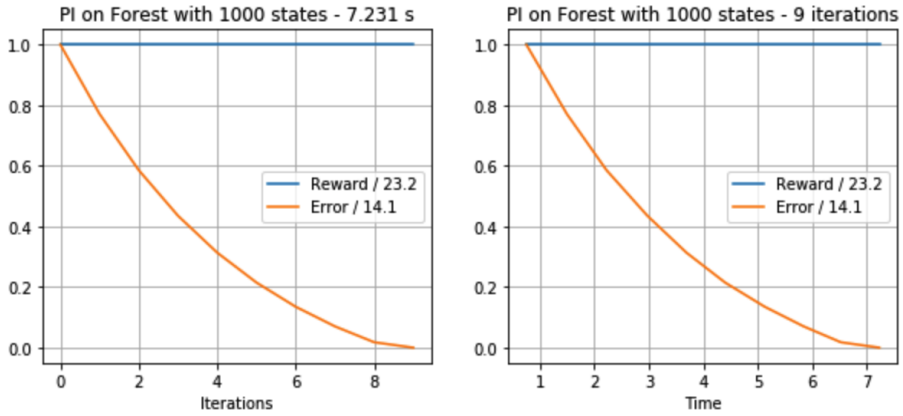

Figure 1: Policy Iteration

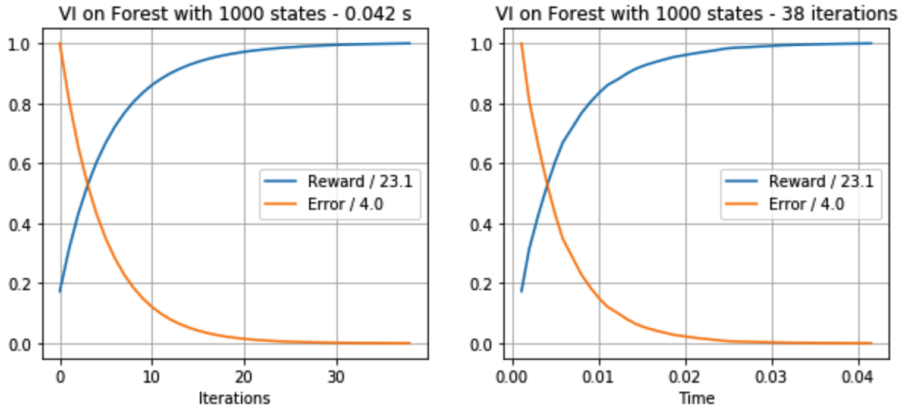

Figure 2: Value Iteration

The curves are nice and smooth and converge quite quickly. We can see how PI converges in a few iterations only, but takes more time. I think this is due to the nature of the algorithm. PI, in its purest form basically solves an MPD at every iteration, which is at least O(n^2), so when the state space size increases, PI becomes slower as we can see in Fig 4 where I compared both methods over various sizes. I think there are bugs in mdptoolbox' metrics, so I wouldn't trust too much the first graph in Fig4, which is why I decided to code the algorithms for MDP2. This is not a library like scikit-learn, used by millions of people. However, my code is designed for Gym's type of MDP's so I couldn't redo the analysis afterwards. For these reasons, I decided to not investigate why the reward is flat for PI.

Fig 3 gives the optimal policies for a Forest with 17 and 31 states. I tried with bigger values, even 500, and they are all the same, ie a 0, then all 1's, and 10 zero's at the end. Now let's see if we can get to that with Q Learning. Fig 5 shows a QLearning simulation for a state space of 20. I had a lot of issues with mdptoolbox Q Learning, so I apologize for the quality of this plot. The policy generated was identical to VI or PI. This problem was quite straightforward and allowed me to prepare for the next one, which is awesome to analyze and tweak.

```
P, R = forest(S=17, r1=4, r2=2, p=0.1, is_sparse=False)
vi, vstats, pi, pstats = JPVIPI(P,R)
```

```
Same policy 57 times in a row when VI converges, we're done!
[0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0]
[0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0]
[0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0]
```

```
P, R = forest(S=30, r1=4, r2=2, p=0.1, is_sparse=False)
vi, vstats, pi, pstats = JPVIPI(P,R)
```

```
Same policy 57 times in a row when VI converges, we're done!
[0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
[0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
```

Figure 3: Optimal policies for Forest with different state spaces

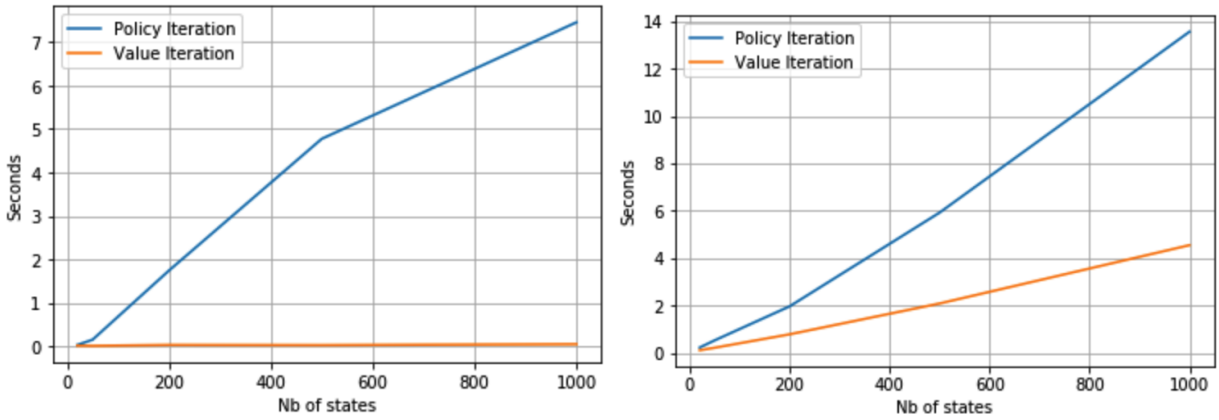

Figure 4: PI vs VI for Forest - mdptoolbox code (left) - my code (right)

1.2 MDP2

1.2.1 PI vs VI

I'd like to start with an 8x8 Lake to lay the ground for the 20x20, but also show a few interesting things. In Fig 6, we can see that there are two holes near the start, in (1,0) and (0,2). Even if the optimal policy manages to go around them, it is certainly not the case with the policy at every iteration.

Here's a few details. I stop the iterations when the maximum value of the difference between the utility function and the previous one goes under a low threshold, typically 1e-5 to 1e-7. All my simulations gave the same policy for VI and PI. I tested it specifically. I did plot the metrics vs iterations and vs time, but as you can see in the graphs below, they are identical. That's because each algorithm does exactly the same thing at every iteration.

The purpose of rewriting the algorithms was to be able to come up with graphs like Fig 7. All the curves have been normalized and the normalizing factor is given in the legend labels. We can see that the utility function decreases quite quickly as expected, in one iteration for VI, ie 0.12s, and about 30 iterations for VI, although also in about 0.12s. The difference in iterations comes from the fact that VI does a lot more. By calculating the intermediate policy, and using it, works very well and 'guides' the algorithm quickly to the optimal policy. To the contrary, VI, in my opinion, fails to use the precious information from the intermediate policy that builds up at each iteration. So I was expecting VI to be slower, but in fact, it iterates a lot more, but the iterations are much shorter, so even with big Lakes as we'll see below, their execution times are very close. I tried to modify the thresholds to try to make one converge before the other, but if we want them to produce the optimal policy, the threshold has to be kept very small (¡1e-6). Then both VI and PI take the same time.

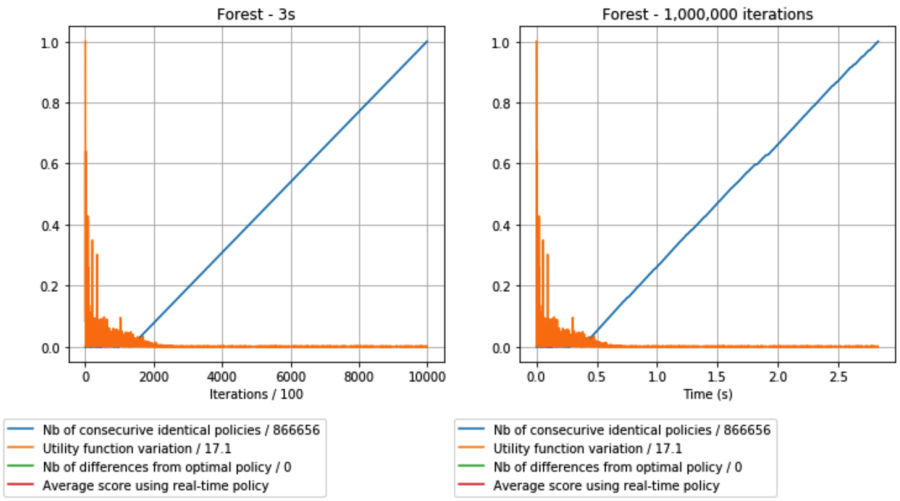

Figure 5: Q Learning simulation on Forest with 20 states

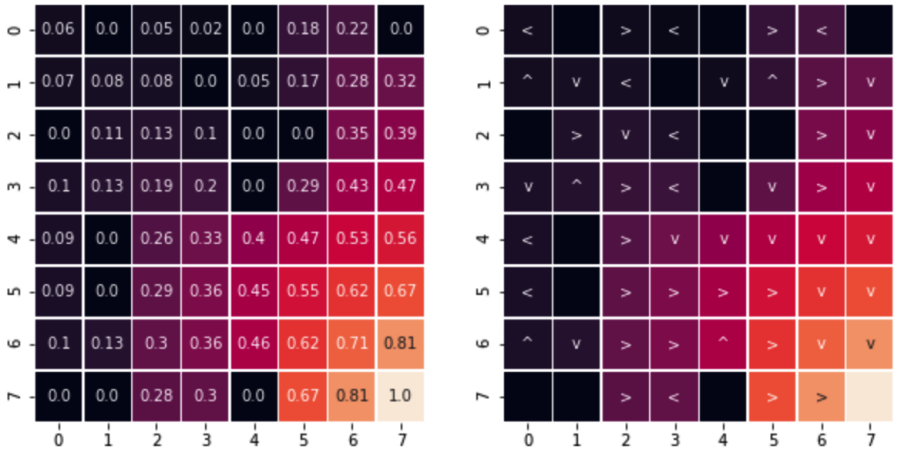

Figure 6: Utility values and optimal policy for 8x8 Frozen lake

I was intrigued by the notion of reaching the optimal policy before the algorithm stops, for obvious reasons (execution time) but also intellectually. I had the idea to create a function that compares two policies and count the states with different optimal actions. I ignored the holes, the goal, and the absorbing state since they don't matter. So, at each iteration, I compare the current policy to the final policy, ie the optimal policy. As you can see, the curve 'Nb of differences from optimal policy' nicely decreases with time or iterations for PI, but it's not the case with VI. I find it very interesting to see how the intermediate policy quickly hits the optimal policy, then leaves, and comes back to it and stays there for 60 iterations (Fig 8). This is a useful tool to see that the threshold value that I used was actually too small.

Another interesting curve is the 'Nb of consecutive identical policies', which also gives a good insight into what the algorithm is doing, and how the policy evolves while the algorithm moves forward. With VI, obviously we exit as soon as the policy doesn't change after one iteration, but for VI, we see bursts of time or iteration where the policy doesn't change while the values do. I can feel here why VI could be superior to VI because it wouldn't 'wait' for the values to change but the intermediate policy 'directs' the algorithm instead.

Last, the 'Average score' curve is the average score achieved with 100 runs at every iteration using the intermediate policy. We can see that it is quite low ~ 0.5 and remains surprisingly constant (with only 10 runs, the curve was much choppier). We can see from Fig 6 that the values near the start are very low due to the presence of the two holes mentioned. So, I did an experiment by removing the hole in (0,2), which had quite

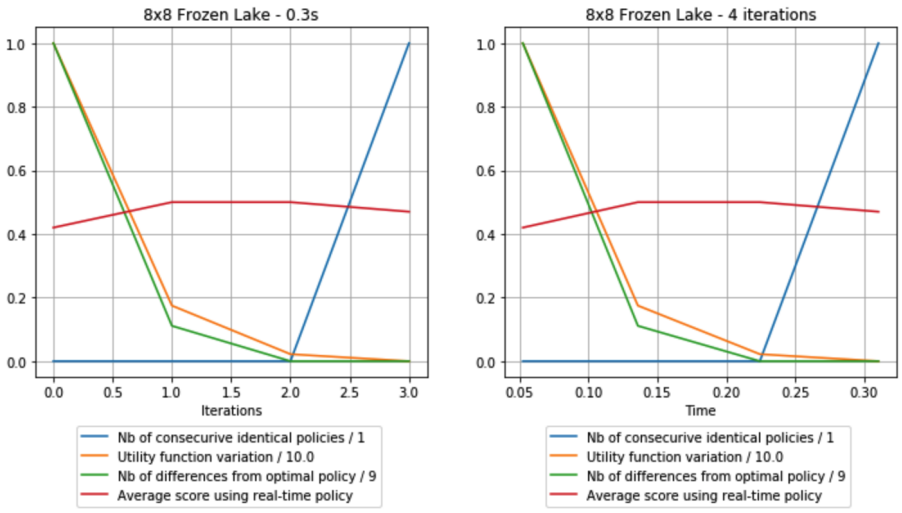

Figure 7: Policy Iteration on a 8x8 Lake

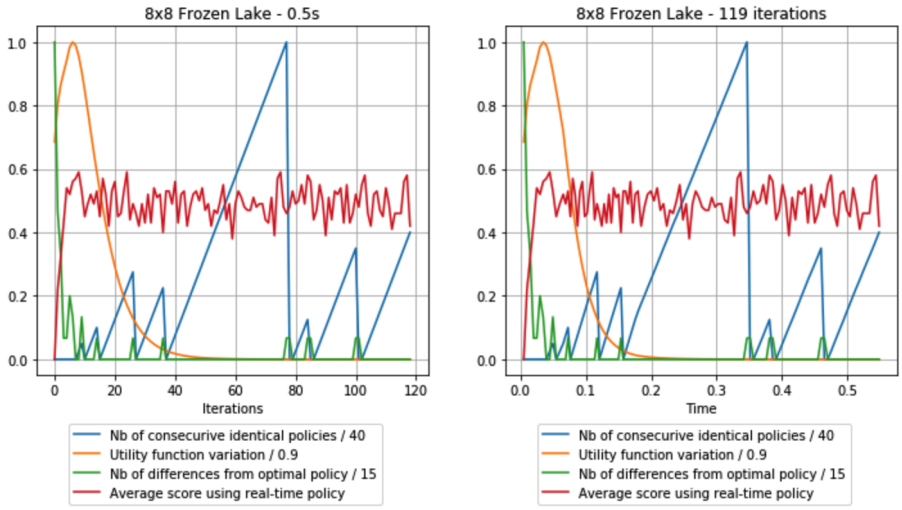

Figure 8: Value Iteration on a 8x8 Lake

a dramatic effect as you can see in Fig 9. The cells around the start are more colored, and the values much higher. As you can see in Fig 10, the average score has increased from 0.5 to about 0.65-0.7. I am not showing the choppier version, with a small average, but of course, the score goes to 1 regularly.

We can see the effect of removing just that one hole. PI looks identical, it didn't get perturbed by it, so I see an advantage here for PI, because VI did get affected by a small change. We see that the difference to the optimal policy quickly settles after 40 iterations, and stays there for 60 iterations. This means that the algorithm keeps fiddling with the values although the optimal policy has been found, but that's because of the small threshold I used. I hope you're not bothered by my using a small Lake but I found it interesting how changing one hole could modify how an algorithm works 'under the hood' and shows the impact on reaching the optimal policy. To be honest, I had to try many maps to do this. Also, since the intermediate policies are not perfect, I did run into cases where they never reach the goal, so I couldn't calculate the average score at every iteration, so I had to pick one that works to produce these graphs.

In the VI figures, we can see the utility of my metrics: when the blue line starts rising and the green line is very low, or zero, we can feel and 'see' the optimal policy taking over, how quickly, how it repeats, with bigger and bigger 'spikes'. To me, it was very helpful to have a better feel of what was going on, but also to compare both algorithms because PI is much quieter and straightforward than VI. Without those metrics, I may have not

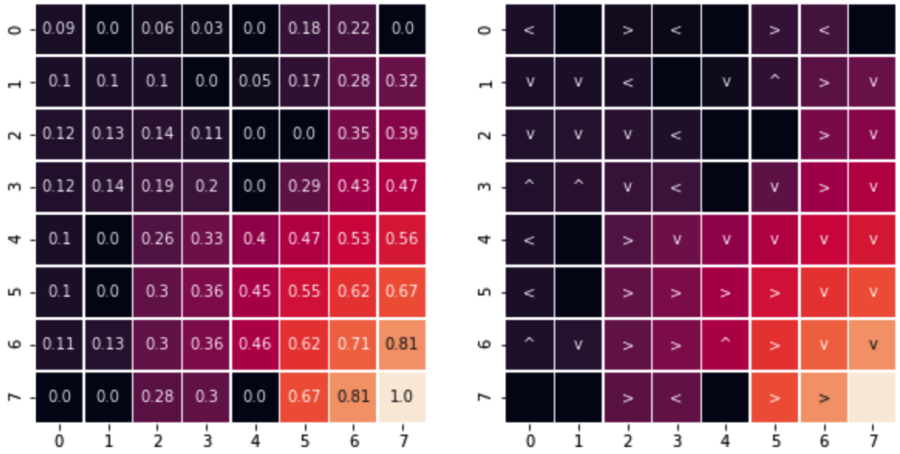

Figure 9: Utility values and optimal policy for 8x8 Frozen lake (modified)

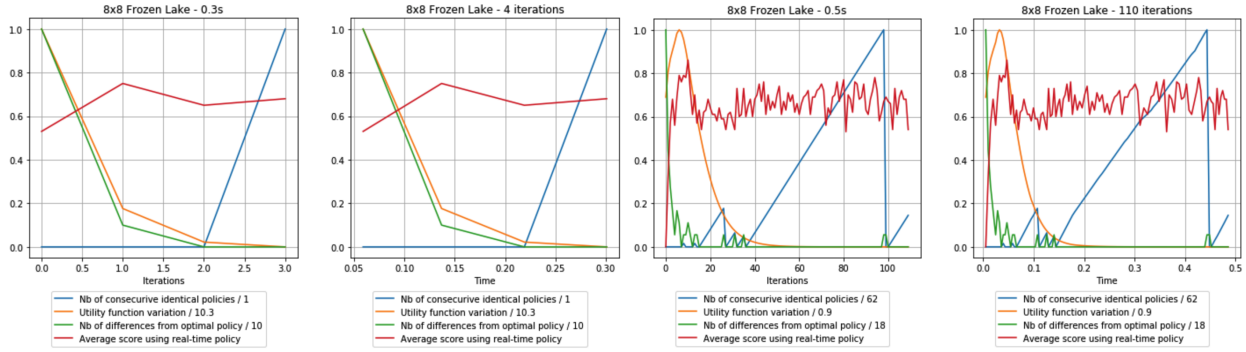

Figure 10: Policy and Value Iteration on a 8x8 Lake (modified)

understood this. I did some simulations to analyze the influence of γ on the reward, which increases with γ. I am not showing it here because it is obvious that, since γ influences how much we look head, like an horizon, low γ
make it more and more difficult to reach the goal.

Now, the 20x20 lake. Fig 18 show the optimal policy (again, identical from VI and PI). The grid with the values has to be big for one to see them, so I put it in the appendix. We can observe how cleverly the algorithms avoids the holes. We can't see it here, but in other maps, the policy crawls against a border at times, playing on the stochasticity of the movement. Fig 12 shows PI which took only 4 seconds to solve this game. The curves are very identical than for the 8x8, except the average score settles much lower, ie 0.25 vs 0.65 before. PI takes only 7 iterations, which is quite impressive. One things to notice is that the average reward now starts at zero, as I was expecting for the 8x8 too. That's because the policy takes a bit more time to start making sense, while, with the 8x8, we get near the optimal policy very quickly so we don't see this slow rise in Fig 7 or Fig 10. I apologize for not showing the cumulative rewards as requested, but I don't feel it is more interesting than the average figure because it shows really what happens at every iteration. We would have probably missed the different types of behavior at the beginning for instance.

Fig 13 shows the simulation for VI. We notice that the green curve (nb of diffs from optimal policy) decreases very quickly but doesn't go to zero before iteration 60, at which point we see big blue spikes, ie the optimal policy is settling as the algorithm nears convergence, in about 4 seconds as well. We can notice that, like previously, the utility function starts rising for a dozen iterations. I could not explain that since I know the convergence proof is based on the notion of contraction. I didn't have time to investigate...

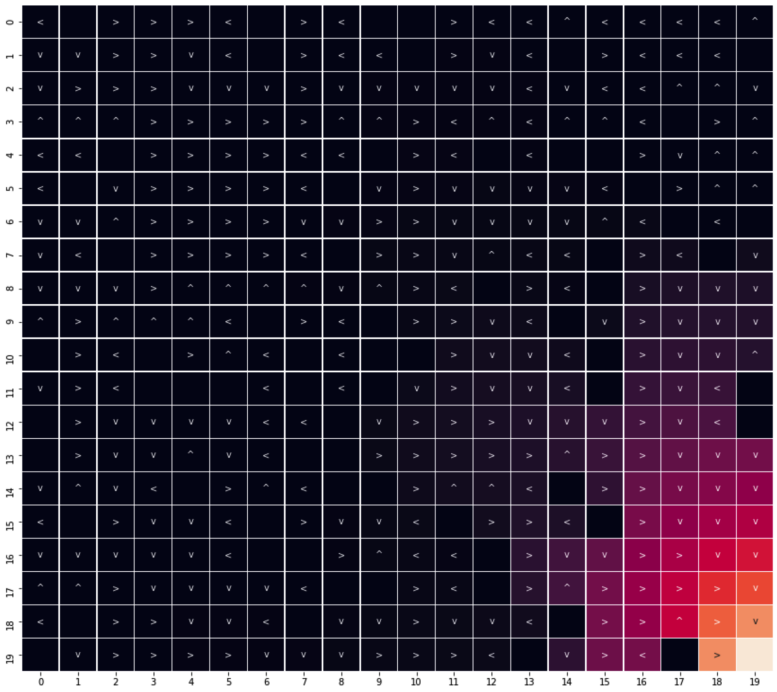

Figure 11: Policy of a 20x20 Frozen Lake

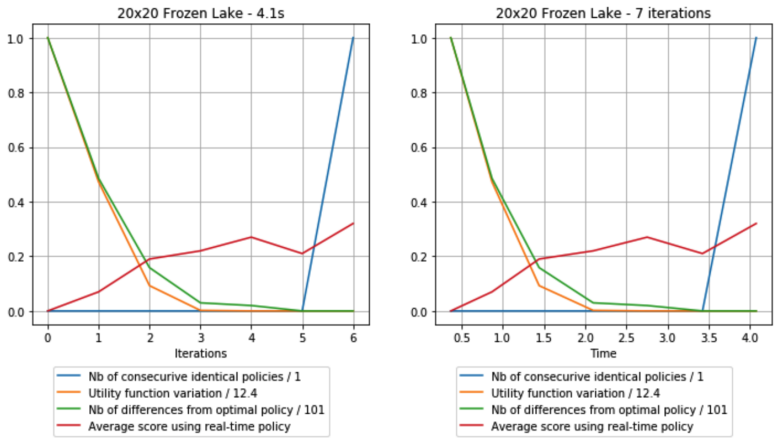

Figure 12: Policy Iteration on a 20x20 Frozen Lake

Fig 14 shows a comparison of execution times for both algorithms for lake sizes up to 30x30. We can see that VI becomes a bit slower when the size increases, and this is due to the extra processing at each step to solve a n x n system of linear equations. We saw this phenomenon more clearly with MDP1 because we could use a much higher n, but here I stopped the size to what I could handle with Q Learning. We know that there are ways to improve PI's speed, like doing bouts of pure VI, but that would have been detrimental for this comparison. I didn't include Q learning because it would be pointless since it is orders of magnitude slower (hours). To keep the policies optimal, I had to decrease the thresholds down to 1e-8, and even so the 25x25 has two errors, and the 30x30 has only one.

1.2.2 Q Learning

Q Learning is so much slower because, first of all, it has to test the 'whole world' and make sense of it. Convergence criteria force us to visit every state, several times. Also, Q Learning follows one path at a time, ie it updates Q values serially when PI and VI use vectorized operations on the whole utility matrix. There must be some parallel implementations out there though, but here, it was slow and painful. I didn't go above 20x20

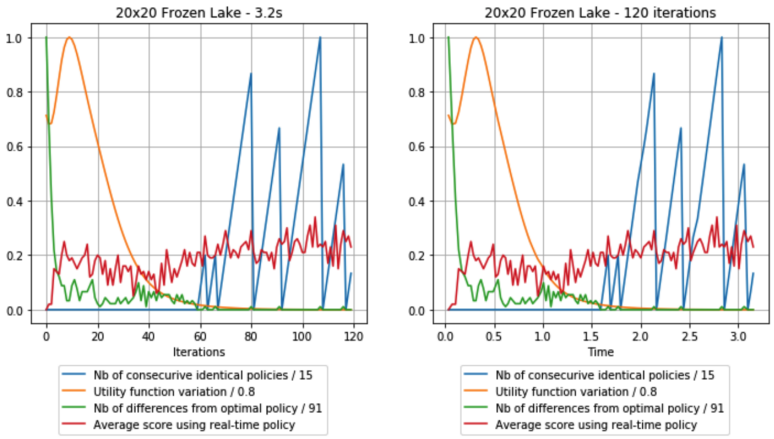

Figure 13: Value Iteration on a 20x20 Frozen Lake

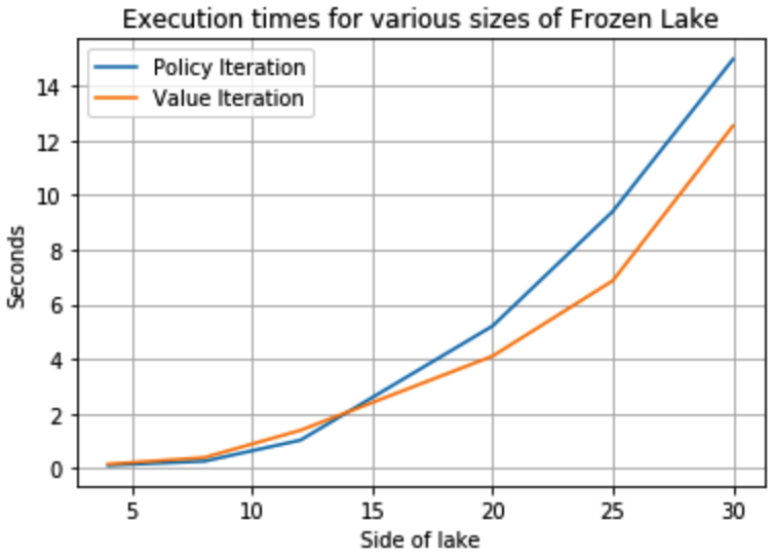

Figure 14: Speed comparison for various lake sizes

because that's really the limit for an assignment like this one with this computer.

I thought long and hard about how to show the differences in convergence with PI and VI. The most important thing to understand was the interplay of gamma, epsilon and the learning rate, and I think curves won't do justice to the complexity. Instead, I decided to show in a table the influence of the various parameters on the behavior of the algorithm. To give a sense of the validity of the optimal policy found, I also logged how many states end up with the wrong policy (vs the known optimal actions), when I could... Here's a few examples.

Table 1 shows three typical experiments among countless ones. The number of parameters, their starting/decay/min values, how they evolve towards the minimum etc is impossible to reflect, so I decide to show the parameters at the end of the convergence since the starting values are less interesting. One word about delta, which is the highest error (sum of absolute values) between the Q matrix after a run and the 10 previous one. When Q learning converges, delta become small, so I use it as a threshold to trigger the decay of the learning rate α and the exponential decay of ϵ. I usually use a threshold of 0.01 to leave enough time to the algorithm to explore (by keeping ϵ high, ie at least above 0.1, so that each of 10 move is random.

The first experiment use a value of γ, 0.96, that I found by trial and error and seems to work. You can see

	γ	α	ϵ	delta	nb visits to all states	Average reward	policy errors	CPU time (s)	Iterations
1	0.96	0.017	0.001	0.04	16	12%	47	2,071	231,658
2	0.96	0.01	0.01	0.019	64	20%	35	10,300	910,700
3	0.98	0.29	0.01	0.2 - 1.5	171	/	/	22,830	1,697,700
4	0.96	0.12	0.01	0.31 - 1.4	132	6%	63	18,506	1,694,100
5	0.96	0.16	0.01	0.8 - 1.7	218	/	/	24,484	2,326,600
6	0.94	0.005	0.012	0.012	220	19%	35	20,475	2,032,700

Table 1: γ, α, ϵ and convergence analysis

in the 3rd experiment that increasing it even by a small margin, 0.02, gives catastrophic results despite visiting all states 171 times and 1.7M iterations, by which time epsilon prevents any exploration, and I stopped the simulation at that point. But the first iteration was not that bad. Not in the final result, but in terms of learning to not let α and ϵ decrease too quickly, before we could visit all the states enough (only 16 times). The average reward and number of policy errors were poor.

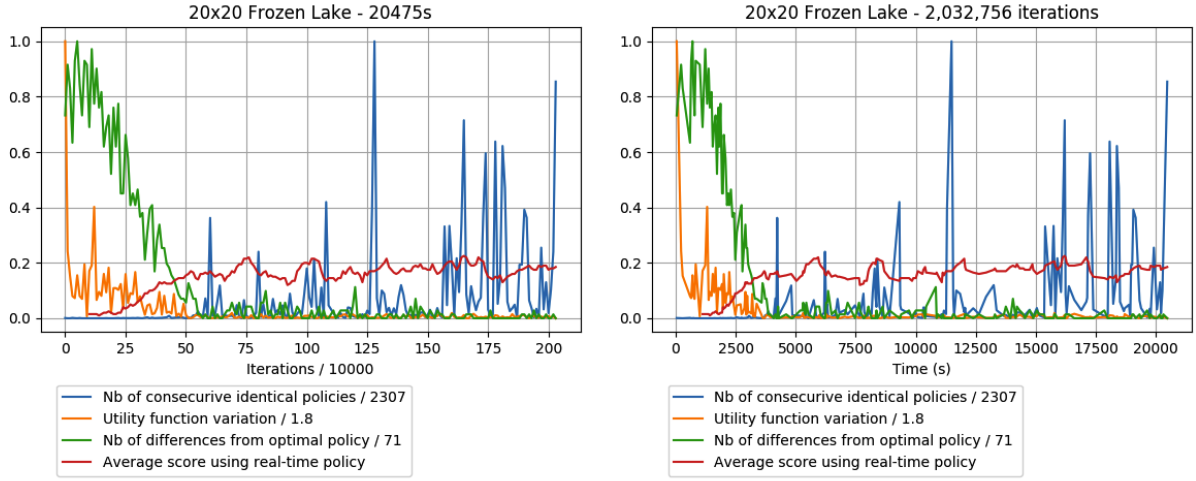

Figure 15: Q Learning performance on a 20x20 Frozen Lake

So, for the next experiment, I didn't let α and ϵ decrease too quickly, and by the time exploration has become impossible, we had visited all states 64 times. The number of policy errors was reduced by 25% and the average reward reached 20% which is almost what we got with PI and VI. This game, again, makes me think about life, where many times, we don't need to be perfect to reach similar performances as those who try much harder. For the 4th experiment, I increased the learning rate's decay rate a bit, but the result was spectacularly bad, which shows one has to hit the right combination of settings. I wanted to try to have a higher learning rate when we stop exploring but it failed. In 6, I went the other way, by lowering γ to 0.94, and making the learning rate very small when ϵ starts blocking exploration, and we got similar results to 2. I tried γ = 0.92, but the result did not improve so the best value is around 0.94-0.96. This confirms the experiments I did with γ in PI and VI. In the end, this horizon is almost a property of the environment, and it was not difficult to determine, ie just a few manual trials (at least here). So I preferred to focus on convergence issues.

Fig 15 shows the convergence curve and metrics for this last experiment. Again, we can clearly see the decrease of the Utility function, the slow(er than before) but steady decline of the green curve, which indicates we get closer to the optimal policy 'like a magnet', which is also evidenced by the spikes in the blue curve. Same behavior as before. Also, notice that the simulation ends with a spike of about 2,000 consecutive identical policies, which is good or bad, depending if we're at or near the optimal pol-

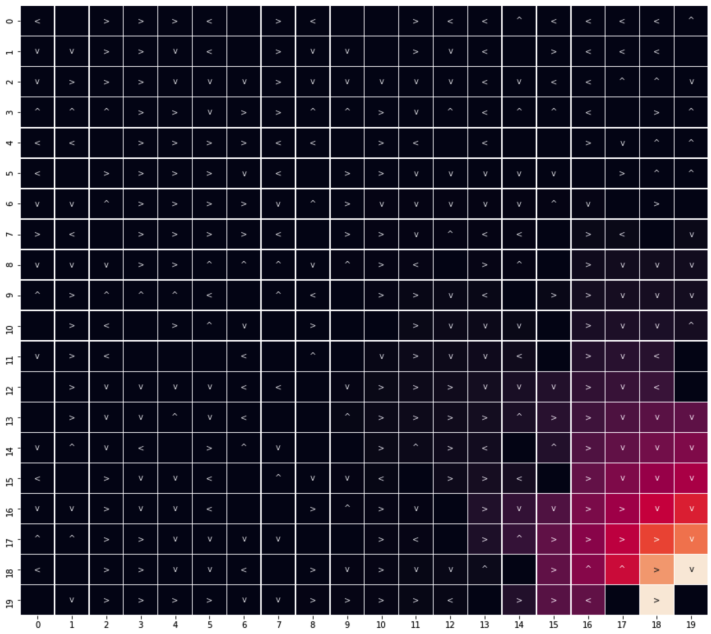

Figure 16: Optimal policy found by Q Learning for experiment 6

icy (green curve low or zero). The green curve is less smooth than for VI and PI, which is normal since the algorithm constantly approaches the optimal policy when it exploits, and gets away from it when it explores.

It is interesting to note that it decreases in almost a linear fashion towards 0, which means the algorithm is moving in the right direction. And then all these little green bumps show that the algorithm tries to 'zeroin' on the optimal policy, and then exploitation begins while the curve stays low. Also, the average score behaves almost identically than with PI and VI. The optimal policy found by QL for experiment 6 is given in Fig **??** and in a bigger format in the Appendix. I didn't try to analyze the errors because we can see that it handles the holes pretty well.

I haven't managed to reach the optimal policy perfectly (best is 91% similarity), but at least I have shown that I understood what to do. Especially, lowering the learning rate over time, which, according to Asmuth, Littman Zinkov [1] is a condition for convergence. Experiments 3 and 4 coudn't even finish for that reason. If I had more time, I'd keep fiddling with the starting values and the decays, to improve the performance. I know it's possible to change the rewards, or put some information in the initial Q(s,a) values, but I am hours from the deadline.

2 Conclusion

This started as an easier assignment, but things can become tricky when one really looks under the hood and tries to understand how these algorithms work and how they differ. I was surprised to see how PI is so competitive vs VI, and converges in a much 'calmer' way. Q Learning on the other hand is a beast that is very hard to tame because it's model-free but also by the high number of combinations of parameters and options. Other algorithms which work on-policy as well, like Sarsa, maybe could do a better job. I am disappointed with my result with Q learning, but at least, I have shown I can analyze what happens and in which direction to go to improve.

3 Appendix

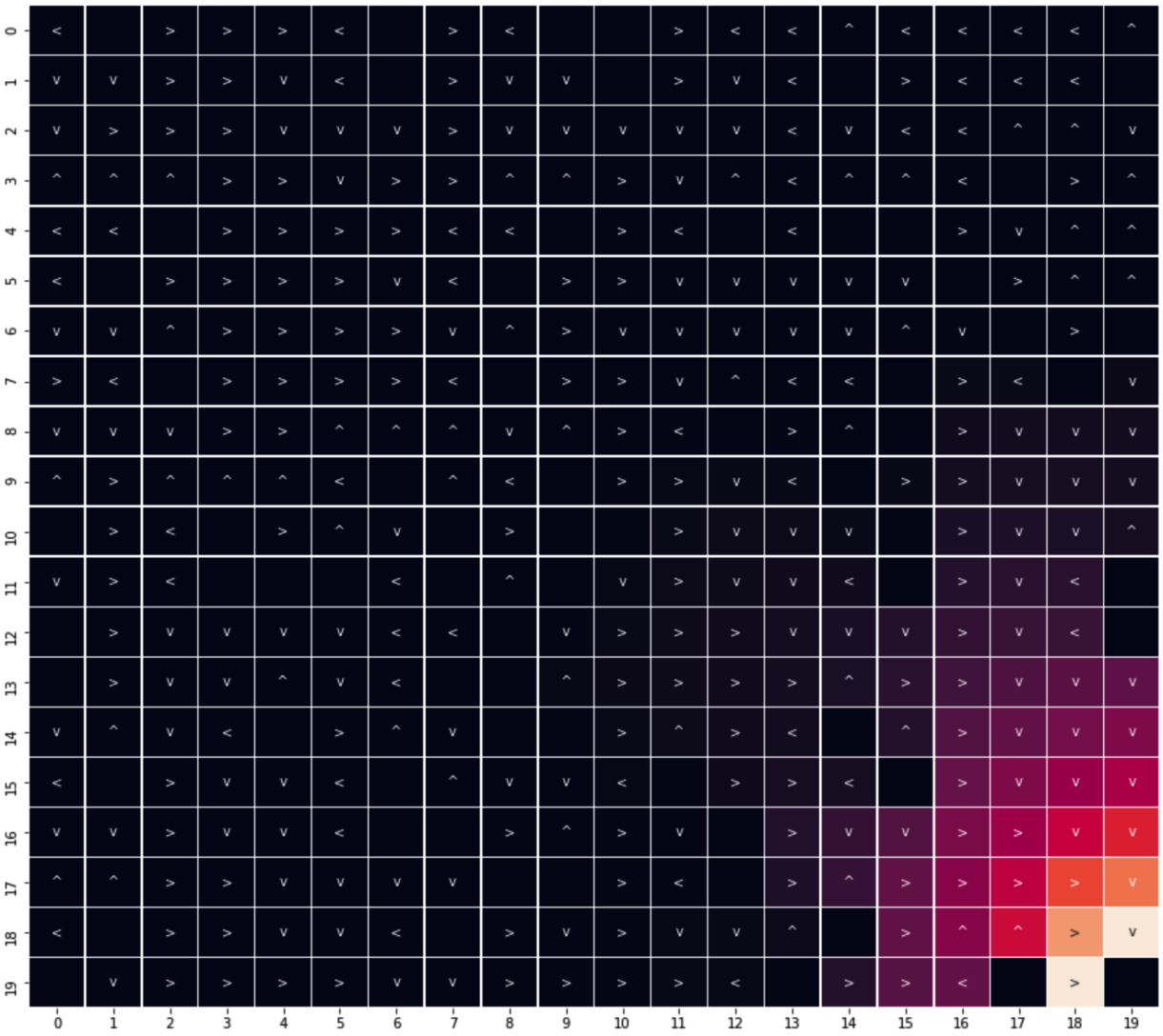

Figure 17: Optimal policy found by Q Learning for experiment 6

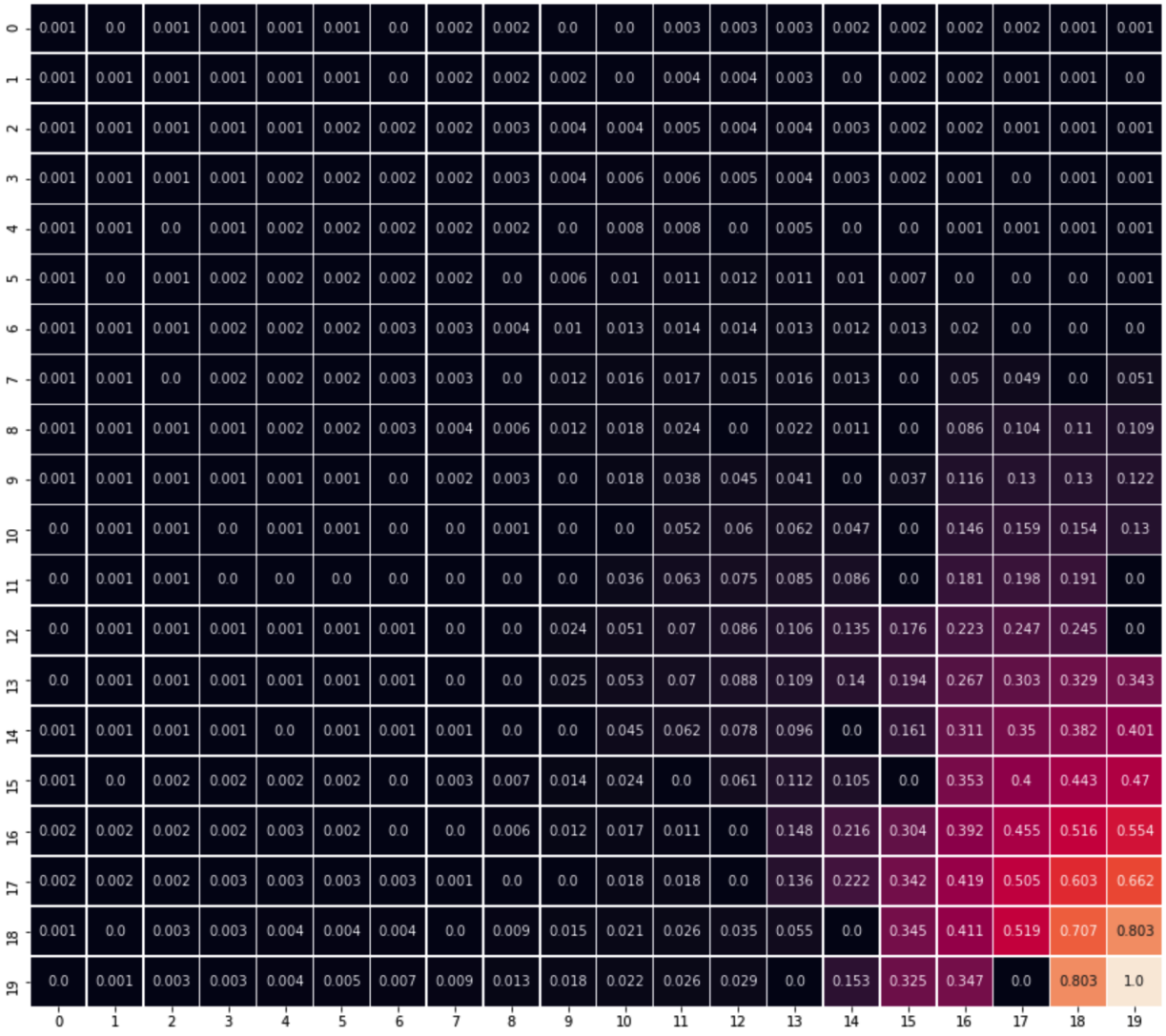

Figure 18: Utility values on a 20x20 Frozen Lake