AI 2 assignment 4

Robin Entjes (S2526883) René Flohil (S2548925)

October 27, 2016

Island Problem

We start off with the policy $\pi(< s1, s2 >, < s2, s3 >, < s3, s3 >)$ and the linear system of equations: $U(s1) = 0 + 0.5 * 0.5 * U(s2), U(s2) = -1 + 0.5 * 0.5 * U(s3), U(s3) = 1 + 0.5 * U(s3)$ according to the Bellman equation: $U(s) = R(s) + \gamma * \max_{a \in A(s)} \Sigma P(s'|s,a) * U(s')$ with a γ of 0.5, P(s'|s,a) of 0.5 and R(s) of 0, -1 and 1 for s1, s2 and s3 respectively.
These equations solve as follows:

```
U(s3) = 1 + 0.5 * U(s3)
1 = 1/U(s3) + 0.5
0 = 1/U(s3) - 0.5
1/U(s3) = 0.5
U(s3) = 2

U(s2) = -1 + 0.5 * 0.5 * U(s3)
U(s2) = -1 + 0.5 * 0.5 * 2
U(s2) = -0.5

U(s1) = 0 + 0.5 * 0.5 * U(s2)
U(s1) = 0 + 0.5 * 0.5 * -0.5
U(s1) = -0.125
```

Now that we have the utilities we can see if the policy changes

```
U(s1) = 0 + 0.5 * max (U(s1), 0.5 * U(s2), 0.5 * U(s3))
      = 0.5 * max (-0.125, -0.25, 1)
      = 0.5 (state = s3)

U(s2) = -1 + 0.5 * max (0.5 * U(s1), U(s2), 0.5 * U(s3))
      = -1 + 0.5 * max (-0.0625, -0.5, 1)
      = -1 + 0.5
      = -0.5 (state = s3)

U(s3) = 1 + 0.5 * max (0.5 * U(s1), 0.5 * U(s2), U(s3))
```

```
      = 1 + 0.5 * max (-0.0625, -0.25, 2)
      = 1 + 1
      = 2 (state = s3)
```

The policy is now $\pi(< s1, s3 >, < s2, s3 >, < s3, s3 >)$ Repeating the same process we take a look if the policy stabilizes

```
U(s1) = 0 + 0.5 * max (U(s1), 0.5 * U(s2), 0.5 * U(s3))
      = 0.5 * max (0.5, -0.25, 1)
      = 0.5 (state = s3)

U(s2) = -1 + 0.5 * max (0.5 * U(s1), U(s2), 0.5 * U(s3))
      = -1 + 0.5 * max (0.25, -0.5, 1)
      = -1 + 0.5
      = -0.5 (state = s3)

U(s3) = 1 + 0.5 * max (0.5 * U(s1), 0.5 * U(s2), U(s3))
      = 1 + 0.5 * max (0.25, -0.25, 2)
      = 1 + 1
      = 2 (state = s3)
```

We see that the policy is still $\pi(< s1, s3 >, < s2, s3 >, < s3, s3 >)$. The policy thus converged.

Comparing your algorithms

Do both algorithms find the same solutions?

If we look at the outcomes of which actions to perform in what state, we see that in the small maze both algorithms gives us the same result. However, if we look at the big maze we see that there is an actual difference. In the case of value iteration, we see that at a certain point one action leads into a wall, in policy iteration we don't encounter this problem.

What is the number of iterations required for each algorithm to find a solution?

	value iteration	policy iteration
Small maze	7	4
Big maze	11	3

We see that in general the policy iterations takes less iterations. Which is even more surprising is the fact that with policy iteration the number of iterations is even a lot lower than the number of iterations for the small maze with value iteration.

What is the time required for each algorithm to find a solution?

	value iteration	policy iteration
Small maze	0.00074	0.00208
Big maze	0.00833	0.02491

To determine the runtime we took the average of 5 runs. We see here immediately that value iteration is much faster. So even though the policy iteration takes less iterations, it is still much slower. This tells us that each iteration in policy iteration takes much more computing.

What is the influence of the discount factor γ on the way both algorithms perform?

To test the influence of γ on the algorithms we use the values 0 to 1.1 with increments of 0.1 in the small maze. Interesting cases are when $\gamma = 0$ and when $\gamma = 1.1$. In the case of $\gamma = 0$ the utilities of all states are equal to their initial reward. When $\gamma = 1.1$ the amount of iterations increase significantly as does the runtime. This is because the discount factor is basically a decay of the utility. When γ is small the utilities decay fast and thus it forces the agent to take the shortest route towards the goal, taking more risks (thus taking the shortest route past the state that has -1 as a reward). When γ is large the utilities don't decay as fast which give the agent the chance to take longer and safer routes. In the case of the small maze the larger route is only taken when $\gamma = 1$ and the utility doesn't decay. Now when $\gamma = 1.1$ the utilities grow larger the longer the agent takes to get to the goal. The agent best strategy is thus to wait infinitely before getting the reward, as the longer the agent waits, the bigger the reward gets. This is also the reason why discount factors are never used.
Interesting to note is that the switch of taking longer routes to avoid states with a reward of -1 does not seem to happen in the big maze even when $\gamma = 1$. This is because the benefit of taking the longer route (and enduring more rewards of -0.04) doesn't weigh up to the risk of going past the state with a reward of -1 and thus taking the shorter route
So the effect of γ on the performance of the algorithm is that as γ gets bigger so does the amount of iterations and the runtime. Next to that a bigger often γ allows for safer and longer routes to be taken. With value iteration we see a growth in runtime and number of iterations as the γ increases. However checking for the same γ-values in policy iteration we see that in this case the discount factor doesn't play a role at all.

Table 1: Effect of γ on the performance of Value Iteration

γ	iterations	runtime (s)
0	1	0.000144
0.1	2	0.000187
0.2	3	0.000374
0.3	4	0.000369
0.4	4	0.000463
0.5	5	0.000498
0.6	5	0.000663
0.7	6	0.000809
0.8	7	0.000921
0.9	7	0.000896
1	9	0.001136
1.1	4281	0.235495

Table 2: Influence of δ on Value Iteration

δ	iterations	runtime (s)
0.0001	10	0.000913
0.0005	9	0.001143
0.001	8	0.001097
0.005	7	0.000804
0.01	7	0.000404
0.05	5	0.000532
0.1	4	0.000250
0.2	4	0.000456
0.3	3	0.000477
0.4	2	0.000144
0.5	2	0.000331

What is the influence of the stop criterion δ in value iteration?

To test the effect of δ on the performance of value iteration we use the following values for δ [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5]. The stop criterion will influence how long the algorithm runs as can be seen in table 2, letting the algorithm iterate more often causes the policy to be more careful. We can see that when $\delta = 0.2$ a switch in risk-taking behaviour occurs. When we go from $\delta = 0.3$ to $\delta = 0.2$ the policy of the bottom left state changes from going past the -1 reward changes to taking the top route with no dangers (in the bottom left state both routes are the same length). The effect of δ is an increase of iterations and an increase of runtime when δ lowers this causes the agent to take safer routes. Somehow the runtime decreases again when $\delta = 0.0001$, but we see no explanation for this.

Final Question

In the real world we don't have this transition functions. However in real-world problems we don't have such transition functions. Reinforcement learning algorithms determine this in a sort of trial and error way. The transition probabilities are initialized randomly. By trying different paths it keeps track of the cost to reach the goal. If the cost is high the transition probabilities are lowered for this path and if the cost is lows the transition probabilities increase. After multiple training iterations it will eventually know the optimal path.