Assignment 1: Dynamic Programming

Hao Chen (s3990788) & Simone de Vos Burchart (s1746995)
Group Number: 88

1 INTRODUCTION

Dynamic programming (DP) is a method of solving complex problems by breaking the problems down into smaller and smaller pieces. It works by finding the optimal solution to the small pieces first, and using those solutions to solve increasingly bigger pieces, until the optimal solution to the entire problem is found. In the following sections, we will dive into some of the theory behind DP, and we will run some experiments with various DP methods.

1.1 Markov Decision Process

A Markov Decision Process (MDP) is a classical formalization of sequential decision making. It involves states, actions that map from states to states, transition probabilities for any given State-Action combination to a next state, and a reward function that assigns a value to each State-Action-NextState combination. In such a process, actions do not only influence immediate rewards, but also future states and hence future rewards. Therefore, there is an element of delayed reward, and the need to weigh immediate rewards versus delayed rewards [1]. This is done through the discounting parameter $\gamma \in [0, 1]$, which, if less than 1, decreases the effect of rewards later in time compared to earlier rewards. If actions don't have any cost, a $\gamma < 1$ will still find the quickest or shortest solution because a longer solution would have lower rewards.

1.2 Generalized Policy Iteration

Given an MDP, the goal is to maximize the rewards by choosing the optimal action in any state, in other words, optimizing the policy. For this, we use Generalized Policy Iteration (GPI), which is the general idea of allowing interaction between the simultaneous processes of policy evaluation and policy improvement. Policy evaluation is the process of making the value function consistent with the current policy, or in other words, computing the value function of a given policy, for example using the Bellman Equation. Policy improvement is the process of defining a new policy that is greedy with respect to the current value function, meaning that at any state it will select the action with the highest value. This new policy is guaranteed to be better than the last one, until after some number of iterations of these two processes it stabilizes and the optimal policy and value function will be found [1].

1.3 Dynamic Programming

Dynamic Programming is a collection of algorithms used to determine optimal policies given an MDP that perfectly models the environment. They are computationally heavy, making them somewhat impractical, but the theory of DP is essential to understanding other methods that try to find optimal policies without assuming that the model is perfect, and using less computation power. DP is usually applied to finite MDPs. The main principle of DP is to use value functions to guide the search for good policies. One way to do this is with GPI as described above. By iterating over a feedback loop between the policy and the value function, the optimal policy is guaranteed to be found.

1.4 Windy GridWorld experiments

We will conduct experiments with 3 different agents in the *Windy GridWorld* environment, shown in Figure 1. This environment consists of a 7x6 square grid where the agent can move horizontally or vertically. In columns 2 and 5 the agent will be pushed up one square with every step, and in columns 3 and 4 two squares up. The agent starts in (0,3) and the goal is to reach (4,3) where it will be rewarded +35 (and it stops), while each step gives a reward of -1. Hence, it needs to find the shortest path to maximize the total reward. The three agents are Policy Iteration Agent, ϵ-Greedy Policy Iteration Agent, and Value Iteration Agent.

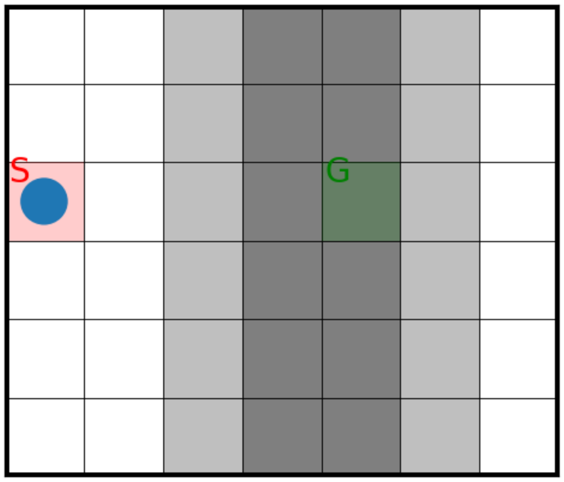

Figure 1: A visualization of the Windy GridWorld. S denotes the starting position, and G the goal.

2 POLICY ITERATION AGENT

The Policy Iteration agent works as described in Section 1.2.

2.1 Methodology

The agent calculates the value of the current policy, and then it computes a greedy improved policy from the obtained values. It repeats this until the policy and value function no longer change.

2.2 Results

As expected, the agent finds the optimal policy, shown in Fig 2. It took 55 seconds to find it, without doing any evaluation iterations.

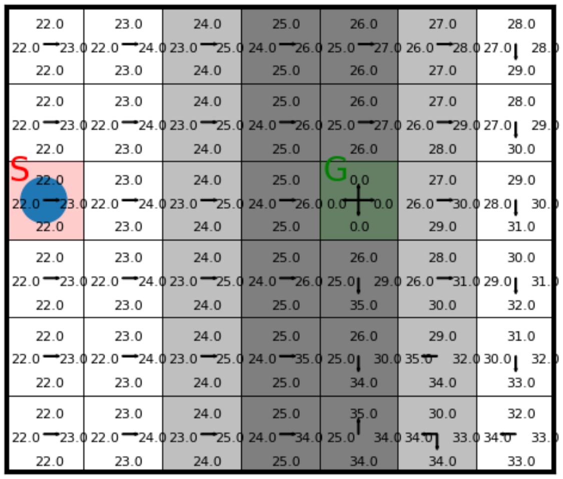

Figure 2: The final, stable result of the Policy Iteration agent. Every state has a Q value for each of the 4 actions.

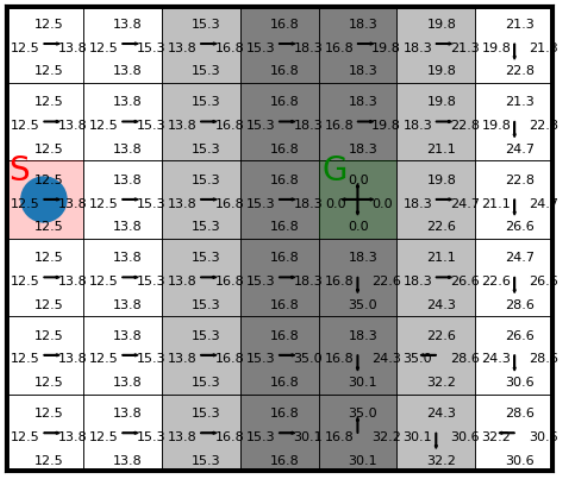

Figure 3: The final, stable result of the ϵ-Greedy Policy Iteration agent. Every state has a Q value for each of the 4 actions.

3 ϵ-GREEDY POLICY ITERATION AGENT

The ϵ-Greedy Policy Iteration agent is almost the same as the previous agent, except for a small difference in defining the new policy in every policy improvement step.

3.1 Methodology

Instead of greedily choosing the best action at any state, it chooses the best action with a probability of $1 - \epsilon$, and the other 3 possible actions with a chance of $\frac{\epsilon}{3}$.

3.2 Results

The policy that the agent ended with is shown in Fig 3. The policy it found is optimal given that the agent moves ϵ-greedily, but it is not necessarily the same as the greedy policy from the previous agent. The Q values of the state/actions are lower, which makes sense since there is a chance of ϵ that a suboptimal action will be taken, which will have a lower reward, so the expected future reward is decreased compared to if only the best action is selected. The agent computed this policy in under 1 second, so it is much faster.

4 VALUE ITERATION AGENT

Value Iteration is a simplified DP algorithm that lets us skip many policy evaluation steps, without losing the guarantee of finding the optimal policy. It stops the policy evaluation after just one update at each state.

4.1 Methodology

We used a discount factor $\gamma = 1.0$, which means that we weigh future rewards the same as current rewards. We also tried different discount factor values: $\gamma = 0.9$ and $\gamma = 0$.

4.2 Results

The Value Iteration agent with $\gamma = 1.0$ found the optimal policy, exactly the same as in Fig 2. Like the Policy Iteration agent, it was guaranteed to find the optimal policy, but the Value Iteration agent found it much faster: in under 1 second. With $\gamma = 1.0$ the Q values for each state/action combination were the same as for the Policy Iteration Agent. With $\gamma = 0.9$, for any state/action that did not immediately lead to the goal, the Q value was lower, because the future rewards were discounted. With $\gamma = 0$, all state/actions had Q value -1, except the final actions to the goal, as with $\gamma = 0$ it looks exclusively at the reward of the current action.

5 DISCUSSION

From our three experiments we can see what we knew to be true in theory: that Policy Iteration and Value Iteration both lead to the optimal policy after some number of iterations. The optimal policy for an agent depends on how the agent acts, e.g. does it act greedily or ϵ-greedily? The ϵ-greedy Policy Iteration agent had lower rewards than the greedy Policy Iteration agent, because at every step there is a chance of taking a non-optimal action with a lower expected future reward. We have seen that Value Iteration is a much more efficient method than Policy Iteration, since it needs far fewer iterations to converge to the optimal policy.

6 CONCLUSION

Dynamic programming is a powerful tool in solving complex problems. The guarantee of finding the optimal solution is very valuable. Although it is computationally heavy, there are aspect that can be made more efficient. We have seen the progression from Policy Iteration to Value Iteration, in which a lot of unnecessary policy evaluation steps were skipped. All in all, dynamic programming is a great method for solving problems where the environment is perfectly modeled by an MDP, and a good foundation for the development of methods that can work with less perfect models.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.