Assignment 1B: Dynamic Programming

Adrien Im (s3984389) & Bence Válint (s3796426)
Group Number: 4

1 INTRODUCTION

Dynamic Programming (DP) is a fundamental approach in Reinforcement Learning (RL) for solving decision-making problems in environments that are fully known. In this report, we will examine DP in the context of the Windy GridWorld environment, as described by Sutton and Barto (2018) [1], and we will compare the two DP algorithms, Policy Iteration and Value Iteration algorithms. We will explore how DP methods solves Markov Decision Processes (MDPs) and analyze their behavior, strengths, and limitations. In the experiment the Windy Grid World is an environment where each steps has a reward of -1 and there is a vertical wind. In columns 2 and 5, the agent is pushed one additional step up, while in columns 3 and 4, the agent is moved up two additional steps.

2 MARKOV DECISION PROCESSES (MDP)

A Markov Decision Process (MDP) gives a framework for maximizing long-term rewards when making decision under uncertainty. MDPs are defined by states $\mathcal{S}$, a set of actions $\mathcal{A}$, a transition function $p(s' \mid s, a)$, a reward function $r(s, a, s')$, and a discount factor $\gamma \in [0, 1]$.

In WindyGridWorld, each state $s \in S$ represents the agent's position on the grid, and the action space is made up of the different actions the agent can make (up, down, left, right). The transition function $p(s' \mid s, a)$ gives the probability of reaching state s' after taking action a in state s. The reward function gives the reward of reaching the state s' from state s with action a.

The discount factor γ determines the importance of future rewards. A higher value of γ gives more importance to long-term rewards, while a lower γ value gives priority to immediate rewards. In the `reward_per_step=0` environment, the value of γ significantly affects the optimal policy. If $\gamma = 0$, only the immediate rewards are considered by the agent. This means that the agent will not move as there is no incentive to do so. As the γ value increases, the agent will be more inclined to find an efficient path to the goal.

3 DYNAMIC PROGRAMMING

Dynamic Programming (DP) is a collection of algorithms that is used to solve MDP problems by computing the best policy when the environment is fully known. In the context of reinforcement learning, one of the applications of dynamic programming is to compute the optimal policy of an environment by evaluating and improving state-value functions or action-value functions. These are called Policy Iteration and Value Iteration and they are two important DP algorithms that will be studied below.

3.1 Policy Iteration

Policy Iteration is made up of two alternating parts: Policy Evaluation and Policy Improvement. Policy Evaluation evaluates iteratively the state-action value V_π for a certain policy. It achieves this using the Bellman expectation equation. Then, using the computed values, Policy Improvement adapts the policy by greedily choosing actions, maximizing the state-action values. This process is repeated until the policy converges to an optimal one.

3.2 Value Iteration

Value Iteration merges the Policy Evaluation and Policy Improvement into one single update using the Bellman optimality equation [1]. This method usually converges faster than Policy Iteration by updating the state-value function $V(s)$ until the maximum update falls below a specific threshold.

DP has the advantage that it guarantees to converge to an optimal solution. It is also efficient in small environments. Therefore, if the state space and action spaces are at a manageable level, DP allows the agent to find the best policy.

However, in order to be able to use DP, we must have full knowledge of the different transition and rewards of the environment. This is something that is mostly not available in real-world problems. As DP requires the full environment to be known, the complexity grows exponentially with state space. This makes it not usable in large state spaces as it consumes excessive resources. Moreover dynamic programming doesn't use state pruning so even unnecessary states will have their values reevaluated every cycle.

3.3 Generalized Policy Iteration (GPI)

Generalized Policy Iteration (GPI) is a process that alternates between Policy Evaluation and Policy Improvement, and allows us to improve the policy faster, and leads to a faster convergence in the policy [1]. Policy evaluation is the process of estimating the value function $V^\pi s$ which is the expected reward when the agent follows the policy π. The second component of GPI is the policy improvement. This process uses a greedy algorithm to change the policy π by choosing an action that maximizes future rewards. In GPI these two processes are repeated until convergence. In essence, dynamic programming is an implementation of GPI, where policy iteration explicitly follows GPI's policy evaluation and improvement cycle. Value iteration on the other hand doesn't explicitly use the GPI policy evaluation and improvement cycle but like stated before it combines these two into one step that is repeated until convergence.

4 EXPERIMENTS AND RESULTS

In our experiments, we applied both Policy Iteration and an ϵ-greedy approach in the Windy GridWorld environment. The state space is represented by the grid, where the wind pushes the agent upwards. There is one start state, labeled S, and one goal state, labeled G. The reward for each state was set to 0, so the discount factor γ plays an important role in allowing the agent to reach the goal.

4.1 Value Iteration Q-values

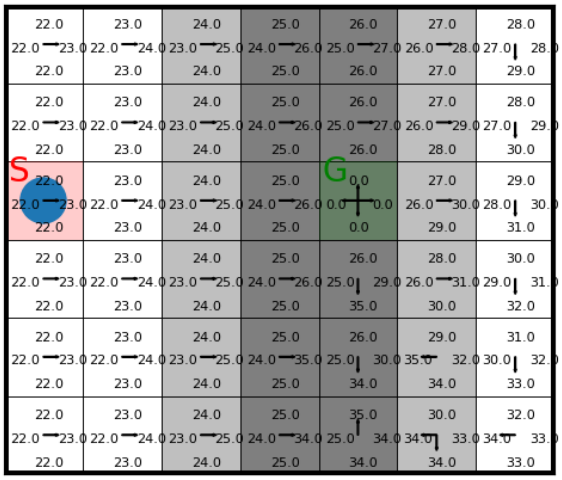

Figure 1: Q-values under Value Iteration

Figure 1 shows the Windy GridWorld environment after converging with Value Iteration. The arrows show the optimal actions. Even though there is −1 reward for each step taken, the γ value of 1 gives the agent an incentive to move towards the goal.

Once converged, Value Iteration gives an optimal policy. From the different grid boxes 4 numbers can be seen each representative of the maximum reward that can be achieved by taking the respective action. The policy is basically following the action with the highest rewards, so in the starting position, the action to go right, which yields the highest expected reward.

4.2 ϵ-Greedy Policy Q-values

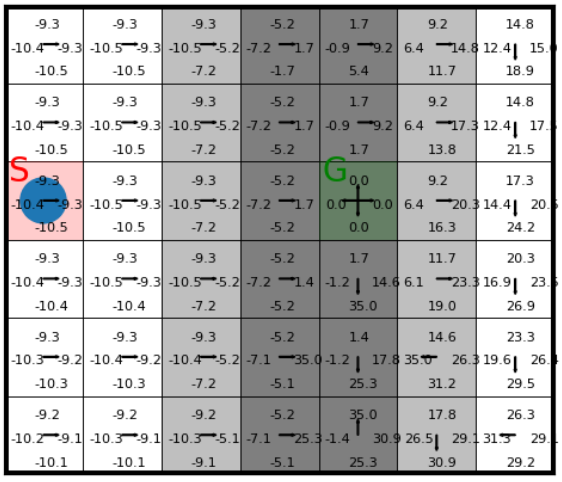

Figure 2: Q-values under an ϵ-greedy policy.

Figure 2 shows the Q values for each action in each cell, and the optimal action at each state. Similarly to the Value Iteration the policy aims to select the best action, however, the ϵ-greedy policy has a certain amount of randomness in the action selection process, given by the value of ϵ. The best action is shown by the highest Q value.

In Figure 2, the state-action values are often below 0, meaning that it would not make sense to take them, since the reward of not moving is 0. This can be explained by the randomness caused by the ϵ, since the randomness might offset the agent leading to more step and more negative rewards for these extra steps, to achieve the goal state.

Comparing Figure 1 and Figure 2, we can see that Value Iteration has significantly greater state-action values compared to the ϵ-greedy policy. As previously mentioned this is due to the randomness in the agents movement caused by the ϵ in the ϵ-greedy policy. On the other hand, the value-iteration agent can simply greedily select the best action and always achieve it in a specific number of steps, yielding a higher expected reward.

5 DISCUSSION

We can derive several insights from our experiment. We can observe that the DP algorithms are able to find optimal policies for the Windy GridWorld environment. We can see the importance of the discount factor γ, especially in the environment in which there is no immediate reward.

We were also able to observe the limitations of DP algorithms. Some further research that can be done is for example using DP concepts and adapt them to be used for partially known environments, instead of fully known environments. Also, more research can be done regarding the impact of γ on the policy robustness.

6 CONCLUSION

Dynamic Programming successfully solves the MDP by making use of its full knowledge of the environment. Using GPI methods such as policy evaluation and policy improvement, we are able to converge to an optimal policy. Despite challenges in applying DP to larger environments as it requires extensive computing power, we can see how it serves as a foundation to more complex reinforcement learning algorithms. Further research about how these DP techniques can be used for complex real-life applications would be interesting to perform.

REFERENCES

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