A2 Mohammad

Introduction

> Write a short report on the problem and the results of your four algorithms.

The purpose of this report is to summarise the findings of Dynamic Programming (DP) and Learning algorithms to a grid-world problem. The DP algorithms observed are Value Iteration and Policy Iteration. The learning algorithms observed are Sarsa Learning and Q-Learning.

Throughout this experiment, it is observed that Dynamic Programming algorithms are consistently better in terms of their optimality/episode. Dynamic Programming algorithms execute faster on average. Lastly, it is observed that Dynamic Programming algorithms are more restrictive in their use case and present more complexities to implement.

Environment

The environment for the grid-world experiment is a 10×10 grid. Each grid tile can be seen as a state and can be one of pit, free-space, wall, or goal.

The rewards for the states are as follows:

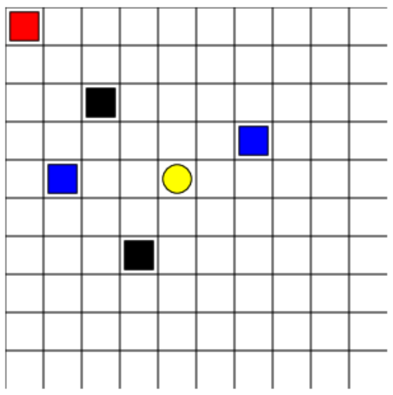
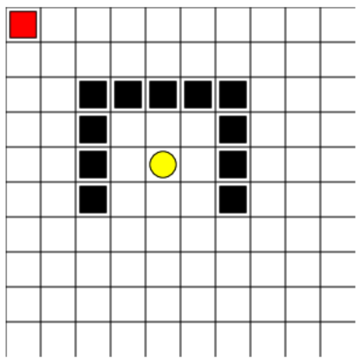
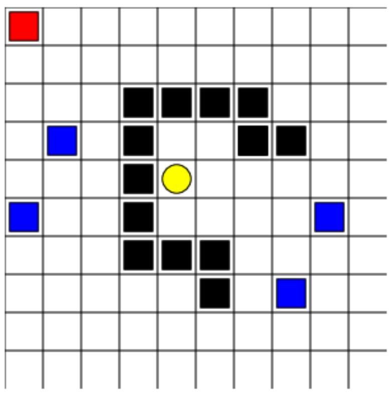
State Rewards

Grid State	Reward
Pit	-10
Wall	-0.3
Free Space	-0.1
Goal	1

An agent cannot go out of bounds from the grid, and if it attempts to do so, it will receive a reward the same as free-space. Additionally, an agent cannot end a state transition into a wall. If an agent attempts to transition into a wall, their state is reset to their last position. Additionally, an episode is terminated if an agent transitions to either a pit of the goal state.

At any state, the agent can attempt to move up, down, right, or left. The environment is deterministic, so a state-action pair (s, a) will always yield the same next state $s\prime$. The probability of taking any equation is equally likely.

Below are the three example grids used in this experiment. The walls are represented as black squares. Pits are seen as blue squares. The yellow circle is the goal state. Lastly, the agent is the red square. All white squares are free-space.

Algorithms

> Describing each algorithm you used, define the states, actions, dynamics. Define the mathematical formulation of your algorithm, show the Bellman updates you use.

In the following section, a brief description of each algorithm is provided. For each algorithm, their states, actions and dynamics will be defined. Additionally, the mathematical formulation of the algorithms along with the Bellman updates will be discussed. The states, actions and dynamics are the same for all algorithms that are defined in the environment section.

Policy Iteration

Policy iteration is a form of dynamic programming. The idea behind policy iteration is that it will improve and evaluate its policies until convergence. Policy iteration always converges to the optimal policy for an MDP as the number of policies is finite.

Policy evaluation updates the agents $v(s)$ to $v_{k+1}(s)$ via the Bellman equation shown below until the changes between successive value functions for all states $s \in S$ are sufficiently small. The policy evaluation process converges to v_π as $k \to \infty$.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s\prime,r} p(s\prime, r|s,a)[r + \gamma v_k(s\prime)]$$

For this implementation of the grid-world, the environment is deterministic so the dynamics equation will always be 1 or 0. An action a at state s will always yield the same next state $s\prime$. This update rule is applied until the difference between v_{k+1} and v_k is smaller than a specified θ value.

Once policy evaluation is complete, the algorithm goes into a state of policy improvement. Policy improvement is based on the *policy improvement theorem*. The theorem states that if for any two deterministic policies π and $\pi\prime$, for all $s \in S$,

$$q_\pi(s, \pi\prime(s)) \geq v_\pi(s),$$

then policy $\pi\prime$ must be as good as policy π. This theorem leads to the following policy update equation,

$$\pi\prime(s) = argmax_a q_\pi \sum_{s\prime, r} p(s\prime, r|s, a)[r + \gamma v_\pi(s\prime)].$$

If $\pi(s) == \pi\prime(s)$ then the iteration phase is said to be stable, and the optimal policy has been found, else the algorithm goes back to policy evaluation.

Value Iteration

Value iteration follows a similar process to Policy Iteration. Value iteration is a special case of policy iteration where policy evaluation is stopped after one sweep of the state space. This is mathematically described as a Bellman equation turned into an update rule,

$$v_{k+1} = max_a \sum_{s\prime, r} p(s\prime, r|s, a)[r + \gamma v_k(s\prime)],$$

for all $s \in S$. For any arbitrary v_0, the sequence of v_k converges to v_*. The update rule is applied until the difference between v_{k+1} and v_k is smaller then a specified θ value.

Sarsa Learning

Sarsa is a learning algorithm, specifically a temporal difference algorithm. This means that the algorithm uses experience to solve the prediction problem for the value functions. After every action, a TD algorithm will make a useful update using the observed reward. This update rule is described mathematically as,

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)],$$

and is applied after every transition from a non-terminal state. Sarsa converges to an optimal policy and action-value function if all state-action pairs are visited an infinite number of times.

Q Learning

Q-Learning is a variant of Sarsa learning, with a slightly different update rule. The update rule is mathematically described as,

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t).$$

By choosing the max_a opposed to the last taken action, the algorithm attempts to directly approximate q_*.

Quantitative Analysis

> Some quantitative analysis of the results, a default plot for comparing all algorithms is given. You can do more plots than this.

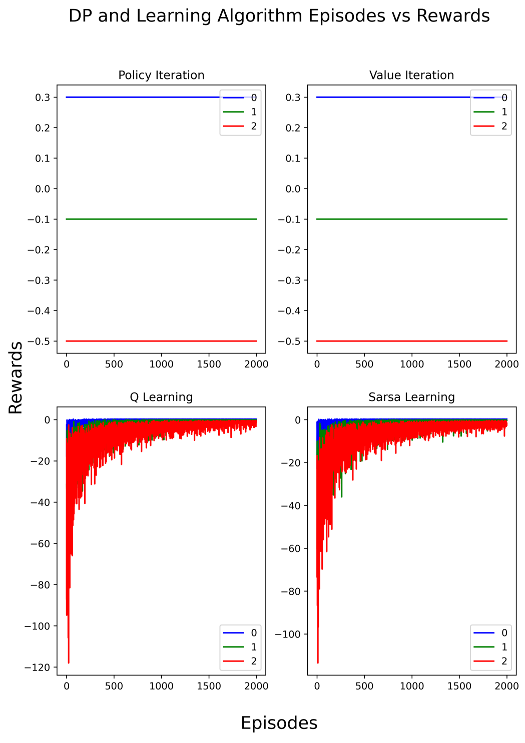
In the following section quantitative analysis of the four algorithms will be discussed. The metrics being analysed are the rate of convergence for each algorithm along with their time to execute.

Convergence

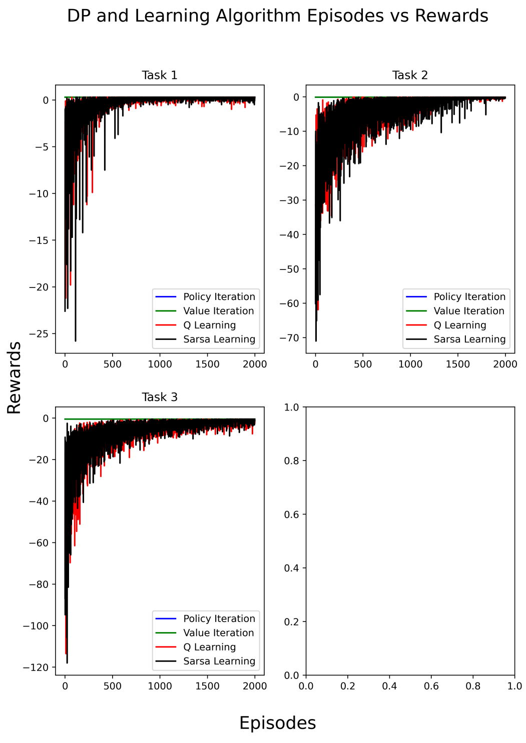
In the diagrams below the observations for each algorithm can be seen and compared by algorithm or by task.

A clear trend shown is the flat line for both Dynamic Programming algorithms, value iteration and policy iteration, when looking at their convergence graph by the task. Additionally, it seems both dynamic programming algorithms converge to the same optimal reward value. It should be noted that the two dynamic programming lines in the graph grouped by the task are barely visible as it is a straight line close to the zero x-axis.

When observing Sarsa and Q-learning, their rate of convergence looks very similar. Both Learning algorithms have very low rewards initially and converge closer to the optimal reward value in a non-linear fashion. When observing the performance of Sarsa and Q-learning grouped by task, it can be seen that Q-learning and Sarsa are very similar in reference to their rate of convergence.



Algorithm performance grouped by algorithm

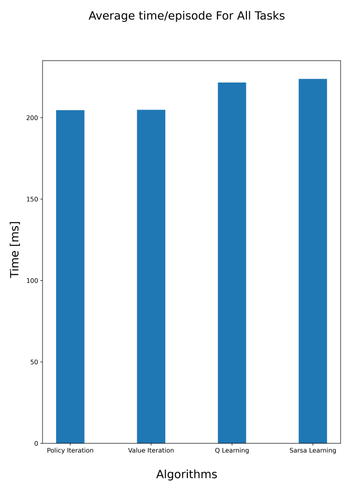


Algorithm performance grouped by task

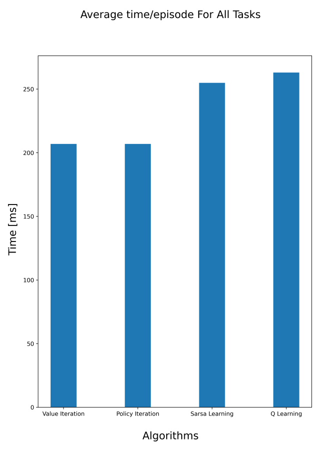
Execution Time

This section of the report will discuss the quantitative analysis for the Dynamic Programming Q-Learning algorithms related to time. The time it takes to finish an episode was recorded for each algorithm for every task. All the times were averaged to remove any small outliers caused by computer hardware or software caching.

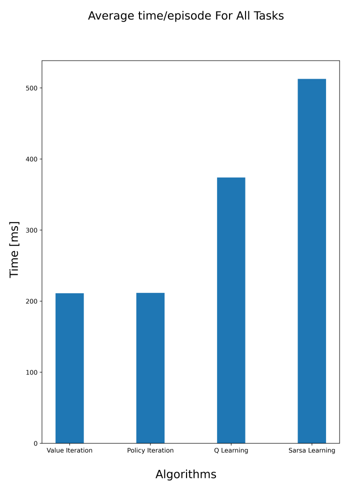
In the figures below, the timing graph is shown for each algorithm for 2000, 500 and 100 episodes. An observed trend is that the two Dynamic Programming algorithms, Policy Iteration and Value Iteration have a constant time close to 200 ms regardless of episode length. Additionally, another trend observed is that the Learning algorithms, Sarsa and Q-learning, converge closer to 200 ms the more episodes there are. Lastly, it is seen that Sarsa Learning takes slightly longer than Q-Learning, however, the difference between the two algorithms reduces/converges to zero as the number of episodes are increased.



Average time/episode over 2000 episodes.



Average time/episode over 500 episodes



Average time/episode for 100 episodes

Qualitative Analysis

> Some qualitative analysis of your observations where one algorithm works well in each case, what you noticed along the way, and explain the differences in performance related to the algorithms.

Implementation

When comparing algorithms for this report, there is a distinct difference between implementing a Dynamic Programming algorithm versus a Learning algorithm. In this report, it is observed that Dynamic Programming presented more challenges and complexities when compared to Learning algorithms.

The source of complexity in the dynamic programming algorithms is having to create an optimal policy from the initialization phase. This process required representing the environment completely, and iterating over the two algorithms, value iteration and policy iteration.

When implementing the learning algorithms, Sarsa learning and q-learning, many of the complexities of the system where removed by having to rely on the environment to inform the agent of good and bad policies. Therefore, removing the environment from the algorithm removes the complexity from the initialization of the algorithm.

Conclusion

The convergence data shows that the two Dynamic Programming algorithms converge from the first episode, whereas the learning algorithms take many episodes to converge to the optimal policy. This is due to the fact that the two Dynamic Programming algorithms create the optimal policy in their initialisation as they have information on the entire environment at once. In contrast, the learning methods need to experience the environment and build up an estimation from those experiences. From the two learning methods seem similar. Therefore, Dynamic Programming algorithms converge a lot faster than Learning algorithms.

In observing the execution time for the four algorithms, it was seen that the Dynamic Programming algorithms all performed close to the same whereas differences were observed for the Learning algorithms. Dynamic Programming algorithms performed the best in time with an average time of 200 ms/episode. In the learning algorithms, Sarsa Learning performed worse when compared to Q-learning, however, the difference between the two converge close to zero as the number of episodes is increased. Additionally, both learning algorithms are seen to be converging close to the optimal time presented by the Dynamic Programming algorithms as the number of episodes is increased. With these observations, Dynamic Programming Algorithms perform better than Learning Algorithms when the number of episodes is small, and learning algorithms approach convergence as the number of episodes is increased.

When comparing the two classes of algorithms (Dynamic Programming and Learning) in terms of their implementation, it is clear that Dynamic Programming cannot be applied to every problem. The reason for this is due to its reliance on understanding the full environment from the initialization phase. In contrast, Learning algorithms, as the name suggests, learn over time what their environment looks like. This allows Learning algorithms more versatile and more applicable to real-world problems. Within each class of algorithms, the complexity between value iteration versus policy iteration and Sarsa learning vs Q-learning where similar.

In conclusion, the two Dynamic Programming methods are ideal when an agents environment is completely known. These methods quickly lead to an optimal policy. However, if the environment is not known then Q-learning should be used. Although both Q-learning and Sarsa Learning converge at the same rate, Q-learning performs faster for smaller number of episodes at no real additional complexity.