Markov Decision Processes

Doan Le

1 INTRODUCTION

The frozen lake problem takes place in a grid world where each space is either frozen ice, a hole, or the goal (Figure 1). The player has four actions available, corresponding to the four directions to move in. When on frozen ice, the player is free to choose a direction; however, the world is non-deterministic in that they can slip on the ice and end up in an adjacent square. This means that a move can have one of three outcomes, and the transition probabilities are split evenly between the three (33%). If a hole is reached, the game ends immediately. If the goal is reached, the player gets a reward of 1, and then the game ends. The player starts in the same position each game, and this position is chosen so that the starting point and the goal are on opposite ends of the board. There is no cost for moving or ending up in a hole, and the only reward is obtained at the goal state.

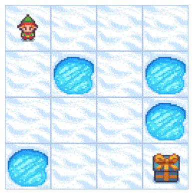

Figure 1 – The frozen lake problem as defined by Open-AI Gym (Source: OpenAI)

The non-deterministic nature, placement of starting and goal states, and lack of movement cost makes this problem interesting. Not only is the problem non-deterministic but the probability of ending in an unintended state (66%) is actually higher than the probability than ending in the intended state (33%). This leads to interesting optimal policies, especially when the player is close to multiple holes. The starting and goal states being on opposite ends of the board requires the player to take a longer route, traversing more of the board. This poses an interesting problem when using a model-free algorithm with a large problem space because it becomes more difficult to find the reward through a sequence of actions. The lack of movement cost is also interesting since there is no motivation to find the shortest path to the reward. With a discount factor close to one, the optimal policy only needs to keep the player out of holes until it eventually finds the goal state.

The forest is a non-grid world problem where states correspond to the age of the forest. The player has the option to either wait for the forest to grow, moving to the next state, or cut the forest. Cutting the forest receives an immediate reward of one, and reverts its age back to zero. If the forest is allowed to reach its maximum age (determined by the problem size), then cutting it receives a reward of two and waiting allows wildlife to thrive, which is worth a reward of four each time this action is chosen. In any state, there is a 10% chance of a forest fire occurring, which reduces the forest age back to zero with no reward.

This problem is interesting since policies can change drastically based on the problem size. In a small forest, it may never be advantageous to cut the forest since the maximum age state can be easily reached with a lower risk of forest fire. However, if the maximum age is much older, this state becomes less attainable, which may change the structure of the optimal policy.

The forest is also interesting because there is no stopping condition to the problem. The player can theoretically reach infinite reward just by continuing long enough. This makes the discount factor play a significant role is determining the optimal policy. This also requires Q-Learning to set a fixed episode length in order to run and the tuning of this parameter may lead to different results.

2 METHODS

Three algorithms are run on two problem sizes of each problem: value iteration, policy iteration, and Q-Learning. The first two are model-based method while the last is a model-free method. Each problem uses discounted rewards and a discount factor of 0.99.

Value iteration is performed with initial values of zero in each state. The algorithm continues until the change in value (value error) is less than 0.001. Policy iteration also uses initial values of zero but converges when the policy no longer changes. If the policy does not converge, the algorithm stops after 20 iterations. This method uses matrix evaluation each iteration to calculate new values.

Q-learning is performed run by initializing Q-values and taking steps through episodes to update the values until they converge. Convergence is met when average Q-value error over the last 10 episodes is less than 0.001. An average is used to prevent early stopping in the case of episodes that end quickly with no changes. On the frozen lake problem, episodes end when either a hole or the goal state is reached. On the forest problem, episodes have a fixed length depending on the size of the problem.

The epsilon parameter is tuned to control the probability of taking a random step rather than a greedy step. The learning rate, alpha, is tuned in order to determine how far to adjust values at each step. Both epsilon and alpha are decayed geometrically by a tuned decay factor until convergence is reached or a minimum limit is reached. For epsilon, this minimum varies based on the problem and problem size. For alpha, the minimum learning rate is 0.001 for all runs so that convergence is reached.

3 FROZEN LAKE

The frozen lake problem is solved on a 10x10 grid and a 20x20 grid. Each grid is generated with the start and goal states on opposite corners and the other spaces having a probability of 0.1 of being a hole.

3.1 Small problem size

3.1.1 MODEL-BASED

The optimal policies found from value and policy iteration are almost identical (Figure 2). In most cases, the algorithm is able to find holes and determines the optimal action is to move away from them. The exception to this is when there is a dense area of holes. In the last row and 4th column, the optimal policy for both algorithms says to move into the neighboring hole on the left. At first this seems non-intuitive; however, considering the transition probabilities of this problem, this would actually me more optimal than moving straight up. If the player moves right, they have a 66% chance of slipping and ending up in a frozen space.

While these two methods result in the same policy, their behaviors are different as shown in their convergence plots (Figure 3). Naturally, the iterative process of value iteration shows a smooth improvement in value error with the number of iterations. The algorithm learns quickly in the beginning, and the gradually approaches negligible error where it terminates upon reaching a certain threshold.

On the other hand, policy iteration converges very quickly after a few iterations. By changing the policy itself rather than the values, it's able to explore different actions much faster. Since it does not update values directly, the value error curve is not as smooth.

The error also gets closer to zero than in value iteration since the convergence criteria is based on whether or not the policy changes in the next iteration. In this case, there are states where multiple

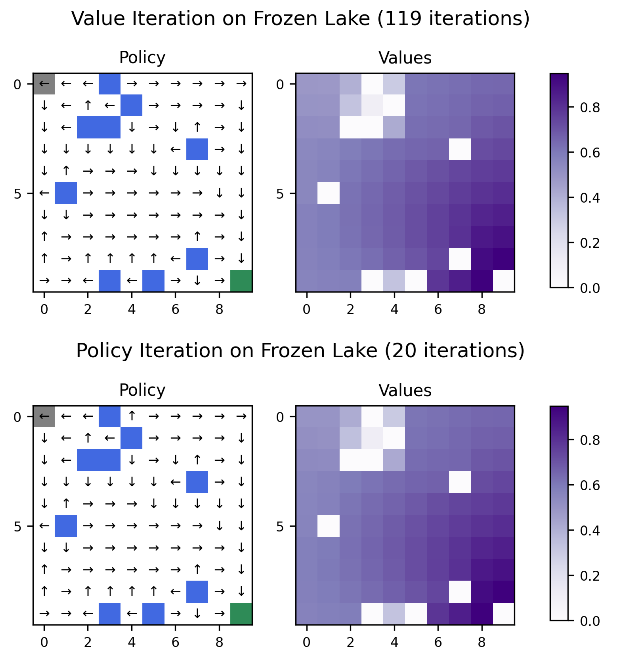

Figure 2 – Optimal policies and values from running value iteration (top) and policy iteration (bottom) on a small frozen lake problem are almost identical.

actions have the same value. For example, in the 0th row and 4th column both moving up and moving to the right have a 33% chance of landing on the space to the right and a 66% chance of ending in a hole. Therefore, when using the Bellman operator to determine the next policy, the result oscillates between moving right and left, not converging on a single optimal policy. This is also a spot where the results from value iteration and policy iteration differ.

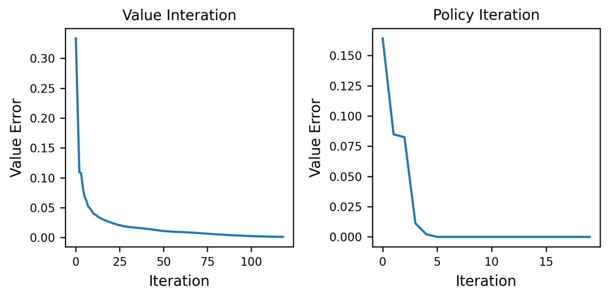

Figure 3 – On the small frozen lake problem, value iteration converges smoothly to a set of optimal values. Policy iteration reaches a point with zero change in value, but iterations continue since the algorithm oscillates between equivalent policies.

3.1.2 Q-LEARNING

As a model-free method, Q-learning performs differently than value iteration and policy iteration. In order to arrive at at the optimal policy, the algorithm

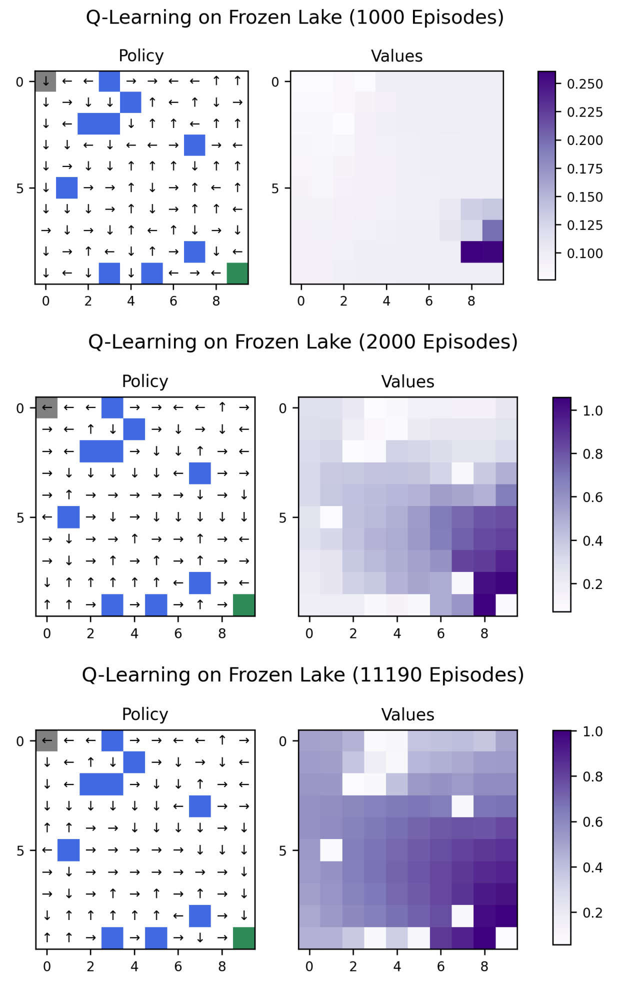

Figure 4 – The learner initially assigns a small positive value to each state and explores the space, recognizing holes and looking for the goal state. Once the goal state is found, it's value is propagated through the grid, leading to a stronger distinction between holes and frozen states that can lead to the goal.

needs to not only find the goal state but also keep track of the holes that it finds. This behaviour was accomplished by careful selection of initial Q-values, learning rate, and epsilon.

For initial Q-value, the best behavior happened when using a small non-zero value (0.1). In early episodes when the reward state is not yet found, the algorithm is able to at least identify holes and make an effort (with certain probability) to avoid them and more efficiently search the space. Once the goal is found, further episodes begin to increase the values of states that could lead to the goal until its value propagates through the grid and back to the initial state (Figure 4).

For learning rate, it was important that the learning rate was still high when the goal state was found

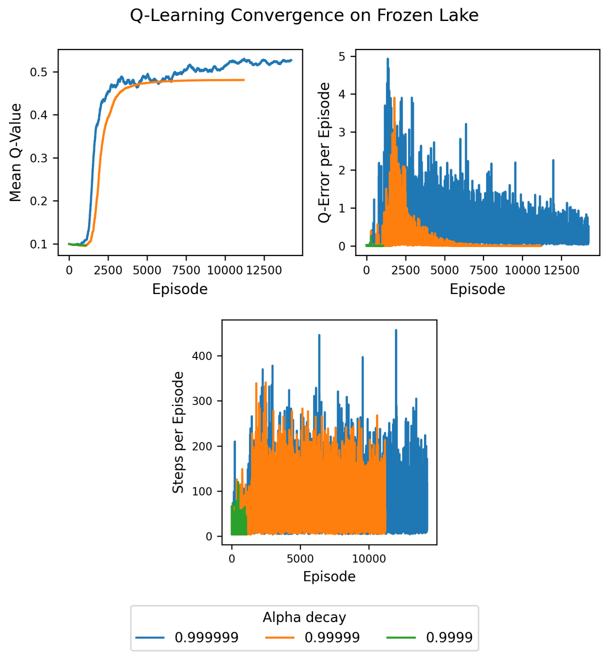

Figure 5 – Q-learning performed at various values for learning rate (alpha) decay shows that there is a balance at 0.99999 where the learner is able to both recognize the goal state and retain information about the holes that it finds.

so that the values can propagate through the grid. However, if the value is too high, the learner seems to "forget" states that may lead to the holes that it found previously. As a result, values fluctuate more than necessary and it takes more episodes to converge. When learning rate is decayed too quickly, Q-Learning converges before it can find the goal state (Figure 12).

For epsilon, the learner needed to behave more randomly in early episodes in order to find the goal state. Higher epsilons also allowed the learner to update values across the grid with the effect of the goal reward. In this problem size, the epsilon decay is not as important since the grid space is small enough to allow effective random exploration.

3.2 Large problem size

3.2.1 MODEL-BASED RL

In the large problem, value and policy iteration also behave similarly. The resulting optimal policies are nearly identical and only differ in states where different actions are tied in value (see the 7th row and 4th column in Figure 6).

The convergence of the two model-based algorithms also have the same similarities and differences as the small problem. Moving to a larger problems space, both algorithms seem to converge slower as the re-

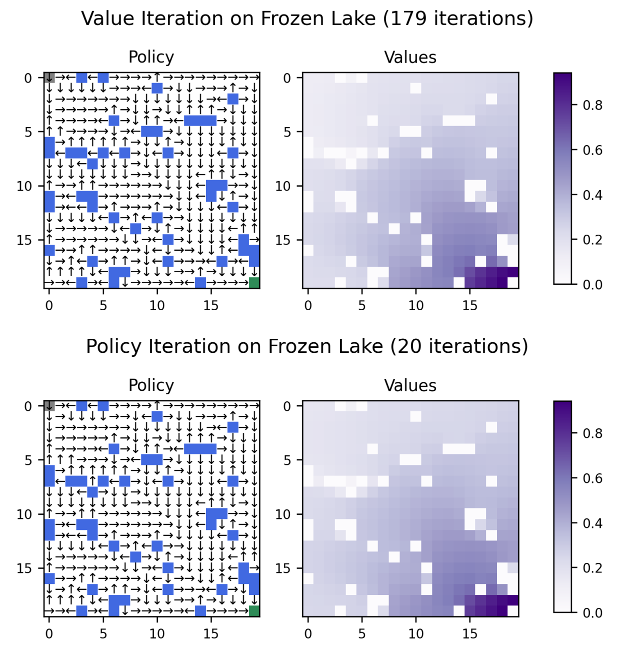

Figure 6 – In the large frozen lake problem, value and policy iteration reach almost identical policies. Differences occur at spaces where different actions are tied.

ward from the goal state takes longer to reach every state in the grid; however, the difference in number of iterations is not detrimental.

Policy iteration appears to scale better with regards to wall clock time (Table 1). Due to states with tied optimal policies, the policy tends to oscillate rather than converge, and the stopping criteria is met when a maximum number of iterations are met. In this case, both the small and large problem spaces run for 20 iterations, and the resulting wall clock time is very similar. On the other hand, value iteration doubles in both time and iterations to meet the stopping criteria.

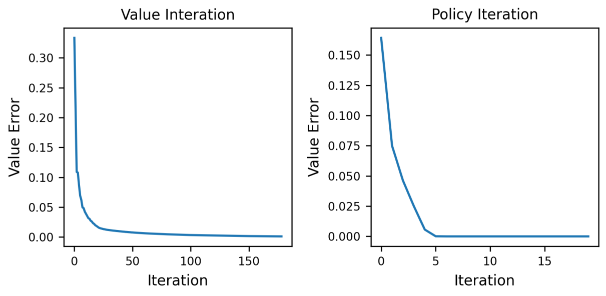

Figure 7 – On the large frozen lake problem, convergence behavior for value and policy iteration is similar to the small problem. Value iteration takes more iterations to reach convergence. Policy iteration reaches zero error slower but still continues due to multiple optimal policies.

Algorithm	Time (small)	Time (large)
Value Iteration	32	71
Policy Iteration	47	263
Q-Learning	17900	254866

Table 1 – The runtimes for each RL method for different sizes of frozen lake are shown in milliseconds. Q-learning is generally slower than the model-based methods and scales poorly to the larger problem size.

3.2.2 Q-LEARNING

The large problem space highlights obstacles encountered with Q-learning. The same tuning is performed as in the small problem. The same initial Q-values (0.1) could be used to encourage the same behavior of finding holes and the goal state. The learning rate behaves in a similar way, but the decay rate needed to be increased since it takes more episodes to find the goal state.

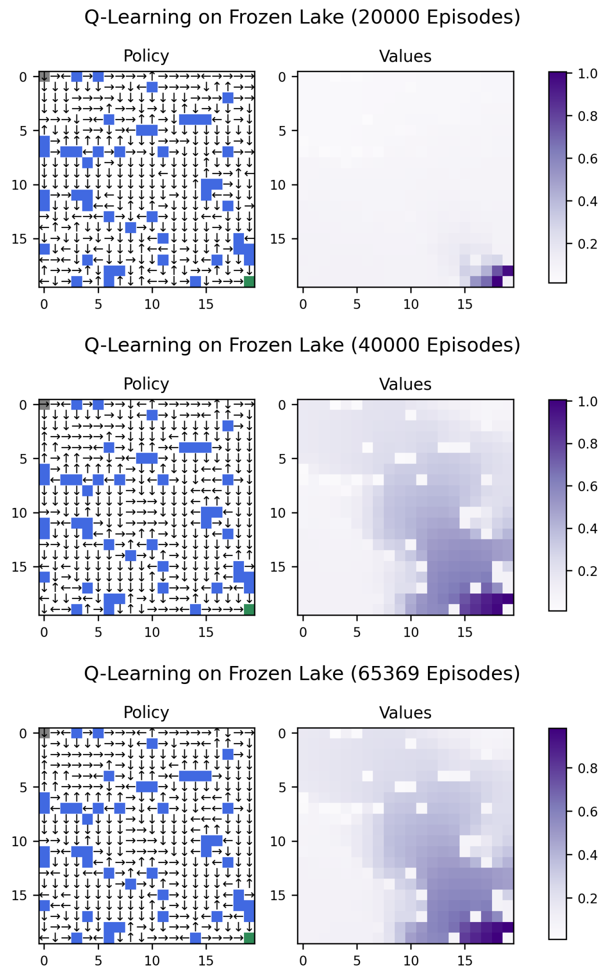

Figure 8 – In the large frozen lake problem, the goal state and holes are found in a similar way to the small problem. However, once the goal is found it becomes more difficult to propagate its value to the rest of the states leaving a corner of the graph unexplored.

Most notably, epsilon plays a larger role in balancing the exploration-exploitation trade-off. In the small problem, epsilon could be kept fairly high and the learner would still eventually find the goal state. In this space, the learner needs to be more greedy to find the goal state in a reasonable number of iterations. Therefore, the reward value seems to propagate along a more isolated path from the starting point to the goal. The bottom left corner of the map is only explored in early episodes when epsilon is high. As epsilon decays to narrow in on the goal, the bottom left is visited less often, and the final values in this region do not get updated (Figure 8).

The number of episodes increases significantly in the larger problems space and highlights the difficulties in using a model-free learner. Unlike value and policy iteration, Q-learning requires many episodes in order to find the reward state and many more to find an optimal policy to get there. The cost of this exploration is seen clearly in the different runtimes shown in Table 1. Since the learner is exploring the world more and more each episode, it naturally needs to take more steps in a larger problem space, and there's also a much lower chance of finding the goal randomly.

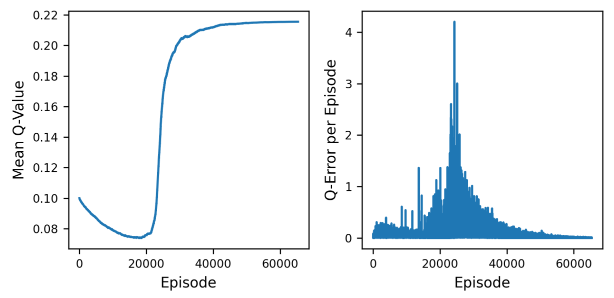

Figure 9 – Q-learning convergence behavior in the large problem space is shown through mean value of all states at the end of each episode (left) and the cumulative error throughout each episode (right).

Since the algorithm learns much slower in the large problem space, its convergence plot displays behavior more prominently (Figure 9). The average value starts out at the initialized 0.1 and decreases as holes are found in the grid. Shortly after the 20000th episode, the learner finds the goal state, and the values rapidly increases throughout the grid until it converges. The total error each episode also supports this narrative. There is some error at the beginning that begins to taper off as most holes are found. Then, the learner finds the goal state and error spikes again until values throughout the grid are updated.

4 FOREST

The forest problem is solved with a maximum age of 20 and 200. The reward for cutting down the forest is zero at the initial age (zero), two at the maximum age, and one in-between. The reward for waiting in the maximum age is two, There's a 10% chance of a forest fire occurring in any state.

4.1 Small problem size

4.1.1 MODEL-BASED RL

Value and policy iteration converge to the same optimal policy for the small forest problem (Figure 10). Most of the time, the forest shouldn't be cut since there's a good chance of reaching the final state and receiving the higher reward for supporting wildlife. The only age when the forest should be cut is the one farthest away from the final state (not including the initial age where cutting has no reward). The values and policy found are highly dependent on the discount factor. A lower discount factor would've prioritized cutting the forest in younger states since the reward from waiting would be weighted less the farther the forest is from the maximum age.

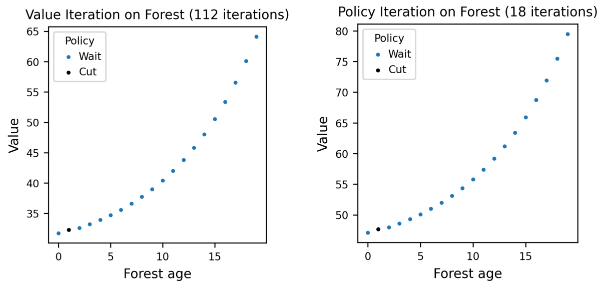

Figure 10 – Value and policy iteration converge to the same optimal policy for the small forest problem.

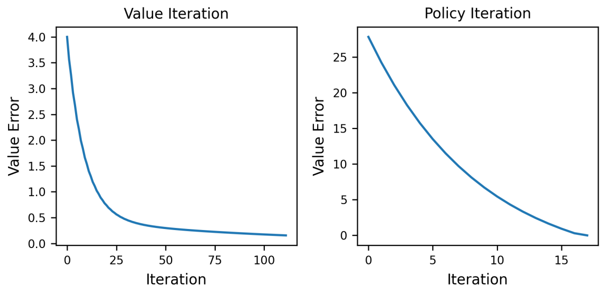

Figure 11 – On the small forest problem, value iteration gradually converges while policy iteration converges in fewer iteration and does not require a long period of minimal changes in value.

The same patterns in convergence for value and policy iteration are seen here (Figure 11). Policy iteration converges in fewer iterations, and unlike in the frozen lake problem, a single optimal policy is reached, allowing it to converge without extra iterations of zero value change.

4.1.2 Q-LEARNING

The best strategy for Q-learning was to run many short episodes that restart in random states in order to explore all of the states. Random exploration is achieved by initializing Q-values with random integers between zero and 100 as well as keeping a constant epsilon of 0.2. The random initialization mimics random behavior in earlier episodes without needing to raise epsilon. Therefore, the algorithm can still be somewhat greedy as learning progresses. Through multiple trials, it was found that episodes of five steps were enough to learn the behavior in a certain range and allowed the learner to restart often in order to explore other areas.

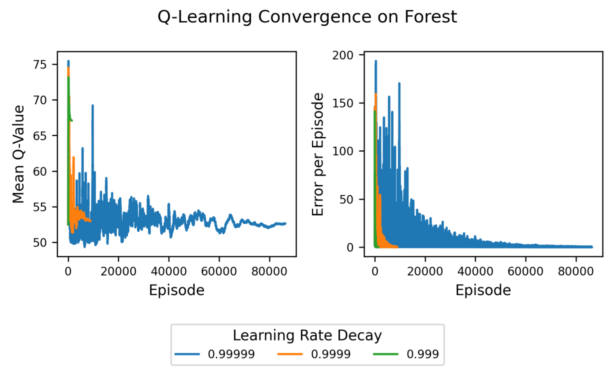

Figure 12 – A learning rate decay factor of 0.9999 allows for both a fast convergence and a correct answer. Learning rates that are too high may still converge to the correct answer but takes more iterations. Learning rates that are too low converge quickly to a non-optimal answer.

Learning rate decay also plays a large role in convergence. When tuning learning rate decay, it was found that a value too low converges quickly to a very different answer since the learner does not get a chance to refine the initial randomness. A learning rate too high not only converges slower but can also forget the initial randomness too quickly and fail to recognize the point where the optimal policy transitions from cutting to waiting (Figure 14).

The final parameters give the convergence curve shown in Figure 14 and the result shown in Figure 13. The early rapid increase in mean Q-value represents the learner finding the reward in the maximum age state. Further changes in mean Q-value refine the value curve. The optimal policy is similar to that of model-based methods.

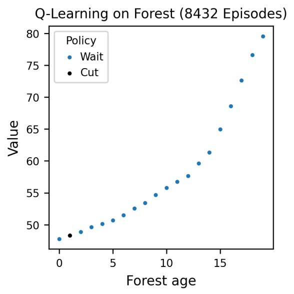

Figure 13 – On the small forest problem, Q-Learning is able to obtain similar results as the model-based methods. The values in the older forest region do not form as smooth as a curve.

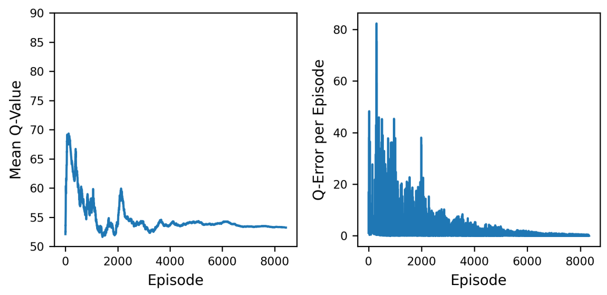

Figure 14 – On the small forest problem, Q-values fluctuate as the algorithm moves the initial random values to points on the curve. The values ultimately converge when the optimal values are found and a small learning rate prevents further changes.

4.2 Large problem size

4.2.1 MODEL-BASED

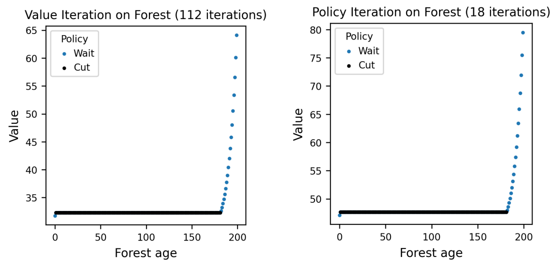

Figure 15 – Value and policy iteration converge to the same optimal policy for the large forest problem. The forest should be cut unless it is close enough to the final state to receive reward from supporting wildlife.

The optimal policy found in the large problem size is drastically different (Figure 15). For a majority of the states cutting is the optimal action since it becomes harder to reach the maximum age without a forest fire. This policy is especially interesting since if a player were too follow the optimal policy, they would end up alternating between the 0^{th} and 1^{st} state forever. This further emphasizes the importance of the discount factor in solving this problem. For the forest to reach maximum age, the discount factor needs to be closer to one to increase the impact of future rewards.

The convergence of value iteration and policy iteration is similar in the small and large problems, converging in the same number of iterations and the same behavior (Figure 16). However, when looking at the run times for both, it appears that the large problem size takes more time to converge (Table 2). It's likely that although the number of iterations stays the same, each iteration takes longer to run since the learner computes values and policies for more states.

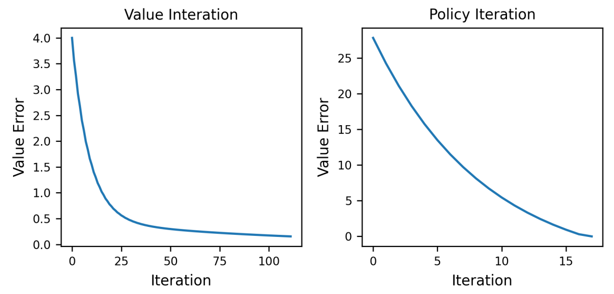

Figure 16 – The large forest problem converges in the same way and in the same number of iterations as the small forest problem.

Algorithm	Time (small)	Time (large)
Value Iteration	4	26
Policy Iteration	23	34
Q-Learning	556	8395

Table 2 – The runtimes for each RL method for different sizes of the forest problem are shown in milliseconds. Q-learning is generally slower than the model-based methods and scales poorly to the larger problem size. Value and policy iteration are slower in the small problem even though they converge in the same number of iterations.

4.2.2 Q-LEARNING

Similar to the large frozen lake problem, the large forest problem showed a need for exploration in early episodes and exploitation in later episodes. The learner does a good job after 5000 episodes of determining the point at which the player should wait rather than cut the forest. After this is found, the Q-values become less random, and the constant, relatively low, epsilon value allows the learner to refine the smooth curve of values found in older ages (Figure 17).

The difference is optimal policy also results in a difference in convergence behavior. In the small problem, Q-values increase and decrease as the shape of the value curve is refined. In the large problem, the value curve generally lies low and flat for most of the state space. Therefore, as the learner steps through the space and inevitable decides to cut the forest, values are gradually decreased. Some small fluctuations occur, but the most drastic change is in earlier episodes.

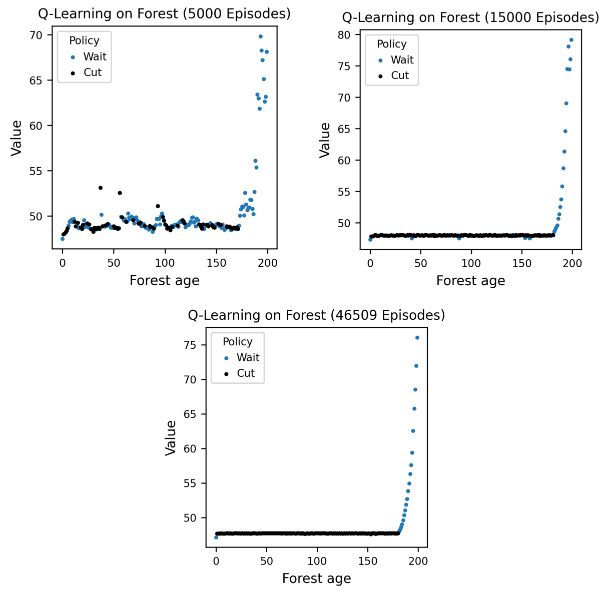

Figure 17 – Q-Learning on the large forest problem gradually adjust values from random numbers to the same result found in model-based methods.

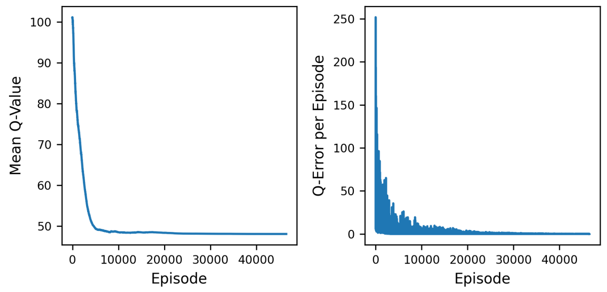

Figure 18 – On the large forest problem, Q-values smoothly decrease since the vast majority of states find cutting to be the optimal policy. The error gradually decreases until the convergence criteria is met.

Along with other parameters, initial learning rate plays a significant role in convergence. A low learning rate can converge to the long answer as the randomness from initialization does not adjust enough to episode results. On the other hand, the learning rate does not need to be too high, which results in quicker changes in the beginning but ultimately takes more iterations to converge.

Q-Learning, again, has the longest runtime out of the three algorithms (Table 2).. It does not scale well to the large problem in runtime nor number of episodes. The difference is not as drastic as in the frozen lake problem. This is perhaps because the forest problem

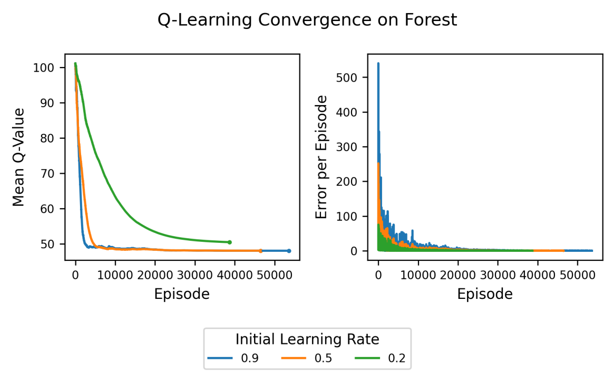

Figure 19 – Q-Learning on the large forest problem shows the importance of the initial learning rate. A learning rate of 0.5 balances correctness and quick convergence.

does not define clear starting and stopping states, allowing Q-learning to start in random states and move on quickly to explore the space more effectively.

5 CONCLUSIONS

In both problems, the model-based algorithms converged quickly to what appeared to be a more accurate result. This benefit is due to the fact that there is no exploration vs. exploration trade-off, and the learner has the advantage of knowing what the rewards are and which states have them from the beginning. Q-learning, as a model-free algorithm, takes longer since it discovers rewards and favorable states as it steps through the world. While the exact reward and transition matrices are not available, the learner can still be effective with enough domain knowledge to form a strategy to efficiently search the space.

The exploration vs. exploration trade-off seen in Q-Learning is seen especially in the larger problem sizes. In general, the learner should explore in early episodes and exploit in later episodes. Often, the learner has to start exploiting before it has explored the entire problem space and regions of the space are left unexplored.