Markov Decision Processes

Andreas Koutras

I. Introduction

In this study, we compare the performance of planning and reinforcement learning (RL) algorithms on two distinct Markov Decision Process (MDP) problems. Planning methods assume that the model of the environment—specifically, the transition and reward matrices—is known, while RL methods take a model-free approach, learning through direct interaction with the environment. We employed value iteration and policy iteration for planning, and Q-learning for reinforcement learning. The two problems selected for evaluation are the Blackjack problem and the discrete Mountain Car problem, both accessible as environments from the Gymnasium library in Python [1].These problems differ significantly in terms of state space size and state transition dynamics, offering a diverse testing ground.

II. MDP Problems and Expectations

Blackjack is a discrete-valued state problem. The interesting aspect is that the the game involves randomness in drawing cards making the transitions between states stochastic. The state space is defined by a tuple with three components: the player's current hand value (card sum), the dealer's visible card, and whether player has a usable ace (an ace that can be counted as either 1 or 11, indicating a soft or hard hand) [2]. An ace is initially treated as usable unless doing so would cause the hand to exceed 21. Therefore, the player's hand value ranges from 4 to 21, representing 18 possible hand states. However, since hands valued at 11 or higher can be soft or hard, the total unique hand states is 29. With the dealer's visible card having 10 possible values, the overall state space consists of $29 \times 10 = 290$ states [2]. The action space is binary: the player can either 'stick' (0) or 'hit' (1). The dealer's actions are deterministic, always standing on a hand value of 17 or higher. Splits and doubling are not allowed. Rewards are provided at the end of each episode, with +1 for a win, -1 for a loss, and 0 for a draw.

The Mountain Car is a continuous state problem with discrete actions [1]. The interesting aspect of this problem is that its state space must be discretized before using the planning and RL algorithms. This gives us the opportunity to investigate how the refinement of the state space discretization affects the algorithms' performance. The environment consists of a car navigating a sinusoidal landscape (Fig. 1) [1]. The state space is defined by two variables: the car's position along the x-axis, ranging from -1.2 to 0.6, and its velocity, ranging from -0.07 to 0.07. Episodes begin with the car randomly placed near the valley (x between -0.6 and -0.4) and at rest. The agent has three possible actions: accelerate left (0), apply no acceleration (1), or accelerate right (-1). The objective is to drive the car to the top of the rightmost hill ($x \geq 0.5$) as quickly as possible. To encourage efficiency, the agent receives a reward of -1 for each timestep [1]. If the car fails to reach the goal within 200 timesteps, the episode ends unsuccessfully. Due to the limited force exerted by the agent compared to gravity, the challenge lies in leveraging the landscape slopes to build sufficient momentum to overcome gravity and ascend the hill. Unlike the Blackjack problem, the Mountain Car environment has deterministic transitions, where each state update is fully determined by the current state and the chosen action.

We expect that in both MDPs, PI and VI will converge to the same optimal police aligned with the theory. However, PI is expected to be more expensive in wall clock time due to the iteration involved in the policy evaluation step in addition to the policy improvement. However, this will potentially help PI converge in fewer iterations than VI. For both problems, we also expect that the optimal policy derived from Q-learning will not surpass those of PI and VI, given that these algorithms have access to the full transition matrix. Nonetheless, due to the smaller state space in the Blackjack problem, we anticipate that Q-learning's policy would closely approximate those of VI and PI. In the Mountain Car problem, we expect that the quality of the optimal policy will depend significantly on the chosen state-space discretization. For both MDPs, we anticipate that balancing the exploration-exploitation trade-off and carefully tuning the learning rate would be crucial to Q-learning's performance.

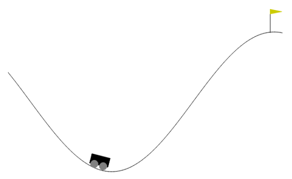

Fig. 1: Mountain car problem environment adopted from Gymnasium [1].

III. Methodology

The environment of the two problems (state space, actions, initialization and state transition) were sourced from the Gymnasium library [1] and were ported into the *bettermdptools* Python library [3]. The latter provides implementations of the planning and RL algorithms used in the study. The wrapper functionality of the Gymnasium was used to port the discretized state space for the Mountain Car and the transition matrix required for the planning algorithms.

To discretize the continuous state space of the Mountain Car problem, the position and velocity ranges were divided into bins. For simplicity, an equal number of bins, N_{bins} was used for both features, resulting in a total of ${N_{bins}}^2$ possible states, indexed from 0 to ${N_{bins}}^2$-1. Given a car's position and velocity in the continuous state space, the corresponding discretized state is determined by mapping to the appropriate bin index. Conversely, the continuous position and velocity for a given discretized state are calculated as the center values of the corresponding bins. The performance of the Mountain Car problem was evaluated using different values of N_{bins}, as will be discussed later.

The Value Iteration (VI) and Policy Iteration (PI) algorithms rely on the transition matrix, which specifies the probability of transitioning to each state given a current state and an action. For the Blackjack problem, computing this transition matrix is complex [2]; hence, we adopted the precomputed transition matrix from *bettermdptools*, which in turn is based on [4]. In contrast, the Mountain Car problem has a straightforward transition matrix due to its deterministic nature: each state-action pair leads to a single subsequent state with a probability of 1, while all other transitions have a probability of 0.

To compare the performance of the three algorithms, we used the expected total return (reward) as the key metric since the main goal of a MDP is to learn a policy that maximizes the expected return. To accurately estimate the expected total return and track its progression during the algorithms' update process, we ran a large number of episodes (1 million for the Blackjack and 10,000 for the Mountain Car problem) and using the policy learned at each VI/PI iteration or Q-learning episode, and then averaged the total returns across these runs. To enhance efficiency in this computation, we leveraged Python's multiprocessing library for parallel execution.

The convergence criteria for each algorithm were defined based on their unique update process. For VI, convergence was assessed by monitoring changes in the state value matrix across iterations. Specifically, the algorithm was considered converged when the maximum difference between corresponding entries in the current and previous state value matrices fell below a predefined threshold. In this study, we used a strict threshold of 1e-10. In PI, the algorithm optimizes the policy directly, therefore, the algorithm was considered converged when the policy array remained unchanged between consecutive iterations, indicating that further updates would not alter the optimal policy. This aligns with the principle that, in PI the policy should improve with each iteration. To ensure convergence detection in both algorithms, a large maximum iteration limit was set, allowing the algorithms to terminate only when the convergence criterion was met. In Q-Learning, the epsilon-greedy strategy introduces randomness in action selection, making it difficult to define a strict convergence threshold given a finite number of episodes. As a result, we assessed convergence by visually monitoring the trend of the expected total return (reward) per learning episode, looking for evidence of stabilization in the reward pattern over time.

To ensure reproducibility and enable meaningful algorithm comparisons, we set fixed random seeds for all stochastic components, including environment initialization, expected return calculations, and action selection in Q-learning.

In Q-learning, the learning rate (alpha) and the exploration-vs-exploitation parameter (epsilon) require careful tuning. For both parameters the implementation in *bettermdptools* uses an exponential-like decay schedule over the learning episodes. Each parameter starts from an initial value (init) at episode zero and decreases to a minimum value (min), which remains constant once reached.The rate of decay is governed by a decay_ratio parameter (ranging from 0 to 1), representing the proportion of total episodes (n_episodes) at which the minimum value is attained. Both alpha and epsilon range between 0 to 1 since alpha determines the fraction of new information incorporated into the Q-value update, while epsilon represents the probability of selecting a random action for exploration. To reduce the number of parameters, we only tuned init_alpha, epsilon_decay_ratio, and min_epsilon, while fixing the remaining decay parameters to init_epsilon=1.0, min_alpha=0.01 and alpha_decay_ratio=0.5, meaning that alpha decays to 0.01 within the first 0.5*n_episodes episodes.

IV. Results for the Blackjack problem

This section presents the results of the VI and PI algorithms applied to each problem.

A. Planning algorithms

In the calculation of the state value, the discount factor (gamma) controls how much weight is given to future rewards relative to immediate rewards. When gamma is close to 1 the agent values future rewards almost as much as immediate rewards, leading to long-term planning. Conversely, when gamma is close to 0 the agent focuses primarily on immediate rewards, ignoring future consequences. The choice of gamma depends on the structure of the reward function (with respect to the state) and the agent's goals (e.g., whether to prioritize short-term gains or long-term rewards). In Blackjack, the reward is given only at end of each episode when the game reaches a terminal state, with no intermediate partial rewards. Therefore, setting gamma=1 ensures that the agent prioritizes the final reward. Additionally, since Blackjack involves a finite number of steps and each hand has a clear endpoint, there is no risk of infinite reward accumulation as in infinite-horizon scenarios. Thus, there is no need to discount future rewards by setting gamma$<$1 to ensure convergence.

Fig. 2 shows the convergence behavior of VI and the impact of different gamma values. At each iteration, we calculated the mean state value across all 290 states, to illustrate the algorithms convergence, and the corresponding expected total return. To accurately estimate the expected total return and capture subtle differences between the algorithms, we found that running 1 million game episodes was necessary. Consistent with our prior analysis, using gamma=1.0 outperformed gamma=0.8, yielding a slightly higher expected return of -0.04227 compared to -0.04229 respectively. This indicates a lower probability of loss with gamma=1.0. The negative expected returns reflect the dealer's inherent advantage over a large number of games. Specifically, simulating 1 million episodes with the policy derived from gamma=1.0 resulted in 47.6% losses, 9.1% draws, and 43.3% wins, with a reward standard deviation of 0.9524.

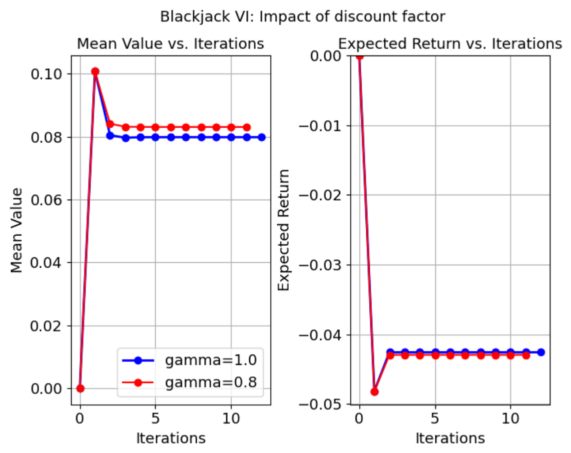

Fig. 2: Convergence of Value Iteration for Blackjack and impact of the discount factor gamma.

Fig. 3 shows the optimal policy map at convergence of VI when gamma=1.0. The policy action for each state is labeled with 'S' for Stand or 'H' for Hit. The rows correspond to the player's hand, and the columns to the dealer's face-up card value. The color map reflects the state value. Notably, the resulting policy is exactly the same as the Basic Strategy for Blackjack [5]! In addition, the state values (expected total returns) show a reasonable pattern. We can see that the highest state values occur when the player's hand totals 21 (i.e., H21 or BJ), as a win is guaranteed in this scenario. Furthermore, for any given player's hand, the state value decreases when the dealer shows an Ace, as the Ace gives the dealer a significant advantage. The lowest state value is observed when the player's hand totals 17 and the dealer shows an Ace, which is reasonable, since the dealer has a higher probability of forming a stronger hand. The optimal policy for gamma=0.8 differs from the Basic Strategy only in the state H16-10 (i.e., the player has a hard 16 and the dealer's face-up card is 10) marked by the red rectangle in Fig. 3. In this case, gamma=0.8 recommends standing, whereas Basic Strategy advises hitting. Standing on a hard 16 when the dealer shows a 10 is a risky move, as the dealer has a higher probability of reaching a stronger hand without busting. This explains why gamma=0.8 achieves a lower expected return than gamma=1.0.

Fig. 4 shows the mean state value and expected return obtained from PI. Since convergence in PI is driven by the policy rather than the state values, the evolution of the mean state value in PI differs from that in VI. However, the expected return follows the same general trend, improving with each iteration, as both VI and PI are guaranteed to converge to the optimal policy. As anticipated, PI converges to exactly the same policy and expected return as VI for both values of

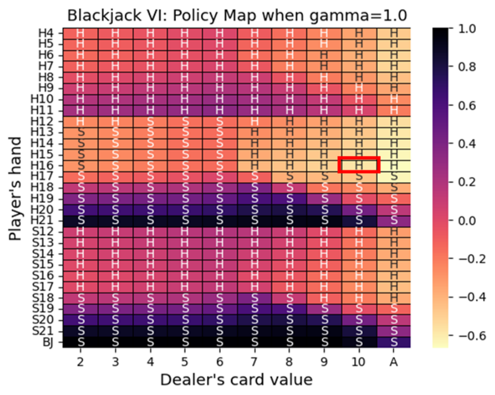

Fig. 3: Optimal policy map for Blackjack derived using Value Iteration.

gamma. It should be noted that PI requires fewer iterations to converge compared to VI. This is likely because, in PI, once the state value is evaluated in the policy evaluation step, the policy update in the policy improvement step leads to significant changes in the policy, requiring fewer iterations to reach the optimal solution. In contrast, VI updates the state value more gradually without explicitly improving the policy at each step, resulting in more iterations to converge to the optimal policy. Despite converging in fewer iterations, PI is more computationally expensive than VI. For gamma=1.0, PI required 0.0495 s of wall-clock time, while VI only required 0.0344 s. This difference is due to the policy evaluation step in PI, which is computationally intensive as it requires multiple iterations to update the state values for each policy iteration.

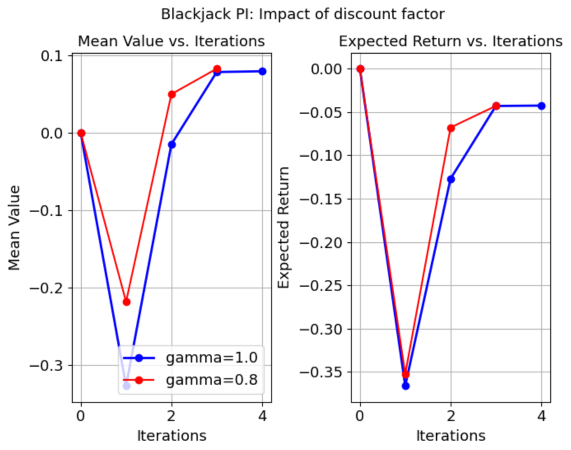

Fig. 4: Convergence of Policy Iteration for Blackjack and impact of the discount factor gamma.

B. *Q-learning algorithm*

Fig. 5 compares the mean state value and expected return from running Q-learning with three different learning rates over 300,000 episodes. To manage the output size and memory usage, we recorded the evolution of state values and policies at intervals of 10,000 episodes. For each recorded policy, the expected return was estimated by running 1 million episodes, consistent with the approach used for the planning algorithms. In this comparison, we varied only the initial value of alpha (init_alpha). The results show that with a larger alpha, the mean state value and expected return exhibit greater fluctuations in the initial stages, as the algorithm prioritizes new information in the Q-value updates, promoting rapid exploration. In contrast, a lower alpha results in smoother, more gradual updates, leading to a more stable progression toward convergence. As the number of learning episodes increases and alpha approaches its minimum value, the performance across different initial alpha settings becomes more similar. Ultimately, the intermediate learning rate (init_alpha=0.5) achieved the highest expected return.

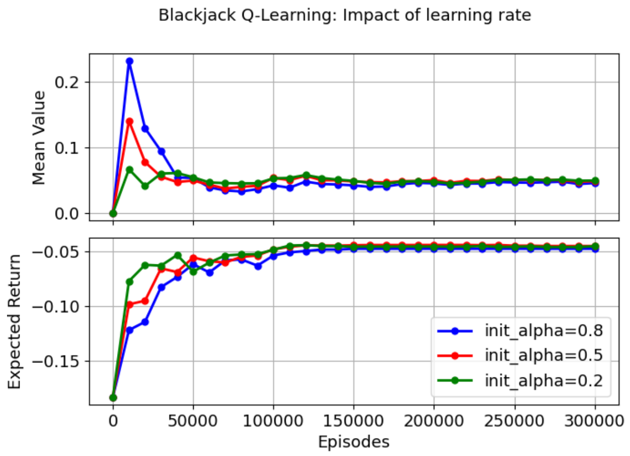

Fig. 5: Impact of learning rate on the convergence of Q-learning in Blackjack.

To demonstrate the epsilon-greedy exploration-exploitation trade-off trade-off, we ran Q-learning with a lower and a higher value of epsilon. Fig. 6 demonstrates the impact of the epsilon parameter on the expected return. The epsilon_decay_ratio was set to 0.9 and 0.1, with min_epsilon fixed at 0.01. In the first case, the residual value is reached within 0.9*300,000 learning episodes, while in the second case, it is reached within just 0.1*300,000 episodes. As shown in the figure, a higher expected return is achieved by promoting greater exploration with a larger epsilon. In contrast, a smaller epsilon the algorithm becomes more biased toward states it has already visited, limiting its ability to explore and learn effectively. On the other hand, the higher degree of exploration with epsilon_decay_ratio=0.9 introduces more variability in the agent's actions and outcomes, resulting in a noisier response. This increased variability can delay convergence over a finite number of episodes.

To tune the parameters of Q-learning, we conducted a grid search to identify the parameter set that yields the highest expected return. The search space covered init_alpha values ranging from 0.2 to 0.8, epsilon_decay_ratio values from 0.5 to 0.9, min_epsilon values from 0.05 to 0.3, and episode counts from 100,000 to 400,000, resulting in a

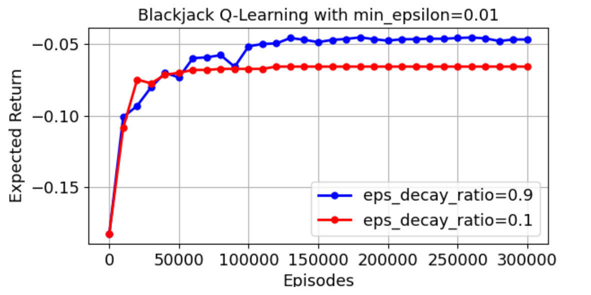

Fig. 6: Impact of epsilon decay on the convergence of Q-learning in Blackjack.

total of 240 different sets of parameters. The optimal set - init_alpha=0.5, epsilon_decay_ratio=0.6, min_epsilon=0.20, and n_episodes=200,000 - achieved an expected return of -0.04253. This result is only slightly higher that the expected return of -0.04227 obtained by VI and PI. Such a slight difference aligns with theoretical expectations, as Q-learning requires an infinite number of visits to each state to converge fully, and the Basic Strategy's expected return captured by VI and PI cannot be surpassed. Additionally, as expected, the model-free Q-learning algorithm comes at a higher computational cost, taking 57.7 seconds to complete 200,000 episodes, compared to 0.0344 and 0.0495 seconds needed for VI and PI to converge respectively. Fig. 7 shows the convergence behavior of the tuned Q-learning algorithm, and Fig. 8 presents the resulting optimal policy. Compared to the Basic Strategy, the Q-learning policy differs in the six states highlighted in the figure.

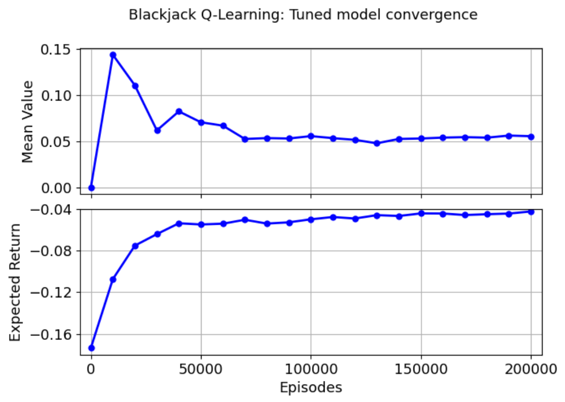

Fig. 7: Convergence behavior of the tuned Q-learning algorithm in Blackjack.

V. Results for the Mountain Car Problem

A. Planning algorithms

Impact of discount factor

Similar to the Blackjack problem, selecting an appropriate discount factor (gamma) is crucial for the Mountain Car problem. In this scenario, the agent receives a reward of -1 for each time step spent without reaching the goal. To motivate the agent to reach the goal quickly, future rewards

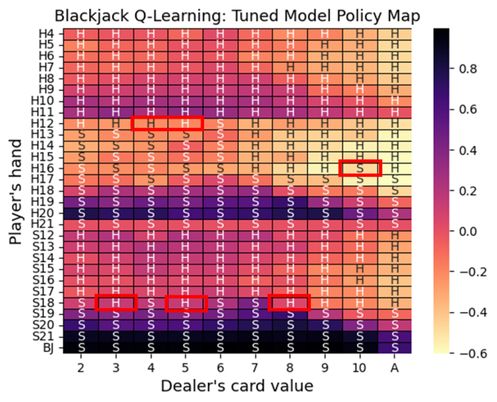

Fig. 8: Optimal policy map for Blackjack derived using the tuned Q-learning algorithm.

should be treated as equally important as immediate rewards, making gamma=1 the most appropriate choice. To demonstrate the impact of using lower gamma values, we ran VI and PI with various gamma values and evaluated the resulting expected total return over 10,000 episodes. This analysis was conducted using a finely discretized state space of 1000×1000 bins spanning the position and velocity ranges.

As shown in Fig. 9, lower discount factors result in lower expected rewards delaying or preventing the car from reaching its goal. For example, setting gamma=0.5 in VI yields an expected reward of -200, indicating that the car fails to reach the goal within 200 steps, causing the episode to terminate. This occurs because a lower discount factor places more emphasis on immediate rewards. Since each time step incurs a negative reward, the agent is not incentivized to prioritize actions that minimize future penalties by reaching the goal quickly. This behavior becomes clearer when examining the car's position history within an episode. Fig. 10 shows the car's position trajectory for an episode where the total reward matches (or is similar to) the expected reward calculated from VI. Unlike higher gamma values that enable the agent to successfully climb the hill, gamma=0.5 results in the car oscillating around a fixed position without making effort to reach the goal. This delay of progress reflects the agent's short-sighted strategy, driven by the prioritization of immediate rewards over long-term gains.

For PI, we found that values of gamma larger than 0.95 made the calculation very computational expensive and were impractical for our analysis. This is likely due to the large state space size making the consideration of future rewards more computationally demanding, the strict convergence criterion on the policy used in PI, and the iteration required by the policy evaluation step. Therefore, we resorted to gamma=0.95 for PI. Conversely, the VI algorithm is able to handle gamma=1.0 being more efficient than PI as described in a previous section

There are some noteworthy observations regarding the performance of the two algorithms. For PI, we found that using gamma values greater than 0.95 made the computation prohibitively expensive and impractical for our analysis. This

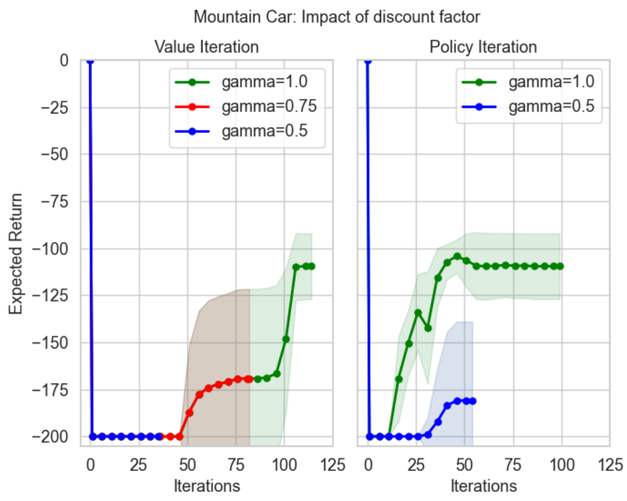

Fig. 9: Impact of the discount on the expected returns of VI and PI for the Mountain Car problem discretized using 1000×1000 bins. The shaded areas represent the range of one standard deviation above and below the mean for each experiment.

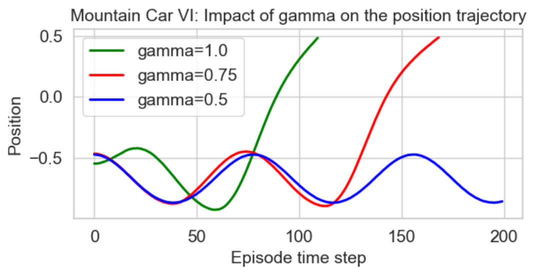

Fig. 10: Car position trajectory under the optimal policy of Value Iteration for different discount factor values. Each trajectory represents an episode where the total reward matches (or is similar to) the expected reward for the corresponding discount factor.

increased complexity is likely due to the large state space, which makes considering future rewards more computationally demanding, coupled with PI's strict convergence criterion and the iterative nature of its policy evaluation step. As a result, we limited gamma to 0.95 for PI. In contrast, VI efficiently handled gamma=1.0, being more efficient than PI, as observed in the Blanckjack problem also. Interestingly though, the final expected return obtained with PI at gamma=0.95 was identical to that of VI at gamma=1.0, indicating that reducing gamma did not impact the final policy. However, for gamma=0.5, PI achieved a higher expected return than VI even though both methods should theoretically converge to the same solution. This is again likely due to PI's stricter convergence criterion, which is applied directly to the policy. Further investigation revealed that VI's results matched those of PI when the convergence tolerance for the state value in VI was tightened from 1e-10 to 1e-20 aligning our result with the theory.

PI required significantly more time to converge than VI, even with gamma reduced to 0.95. Specifically PI took 7302.9 seconds to converge at gamma=0.95, compared to 774.9 seconds - nearly 10 times faster - at gamma=0.5. Meanwhile, VI was completed in just 396.2, 283.8, and 121.8 seconds for gamma values of 1.0, 0.75, and 0.5, respectively. This highlights VI's superior efficiency, particularly for higher discount factors compared to PI.

Impact of state space discretization

To evaluate how state space discretization affects the performance of planning algorithms, we solved the Mountain Car problem using two additional refinement levels—200×200 and 500×500 bins across the position and velocity ranges—alongside the previously analyzed 1000×1000 state space. As shown in Fig. 11, the expected return decreases as the discretization becomes coarser. This result is intuitive, as wider state bins may prevent the car from transitioning to the neighboring state under its current velocity. Additionally, coarser discretization is more sensitive to the randomized environment initialization. Depending on the initial state, the car may not be able to escape its starting position if the bins are too wide. Both algorithms converged to the same expected return regardless of the refinement. Both VI and PI algorithms converged to the same expected return across all refinement levels, demonstrating their consistency. However, PI required fewer iterations to converge compared to VI, consistent with observations from the Blackjack MDP. Despite this, PI was significantly more computationally expensive in terms of wall clock time, taking 182.2, 1617.3, and 7302.9 seconds for the 200×200, 500×500, and 1000×1000 grids, respectively. In contrast, VI completed in just 17.0, 100.5, and 396.2 seconds for the corresponding grid sizes, highlighting its efficiency.

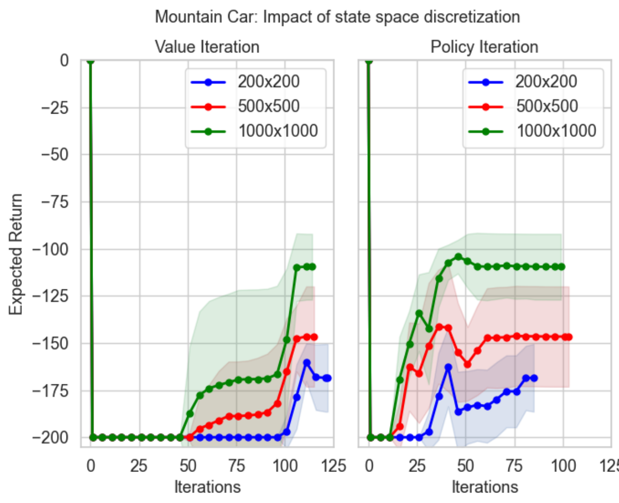

Fig. 11: Impact of the state space discretization on the expected returns of VI and PI for the Mountain Car problem.

Optimal policy

The state space discretization of 1000×1000 with gamma=1.0 (or 0.95 for PI) yielded by far the highest expected reward. Fig. 12 shows heatmaps of the final state values and optimal policy actions across the grid. It can be observed that the lowest state values occur when the car is near the valley (position -0.5236) and has small (or zero) velocity. This is indeed the least favorable state to be in terms of expected rewards. In contrast, the highest state values occur when the car is near or at the target position (+0.50) with positive velocity. The highest state values occur when the car is near (or at) the target position of +0.50 having positive velocity. States near +0.50 but with negative velocities have lower values, as the car is close to the goal but moving downhill. A more promising scenario is when the car climbs the left slope (negative velocity); although it will hit the left boundary, it gains an opportunity to develop positive velocity while descending, propelling it closer to the goal.

The policy map aligns with the state value map. We can see that when the agent is at a state near the visible curved boundary separating higher and lower state values, it should move toward the region with higher state values. Indeed the policy dictates accelerating to the right (action 2), as indicated by the dark regions in the policy map. The zebra-like patterns observed in the policy map reflect subtle contours in the state value map, highlighting the fine variations in optimal actions across adjacent states.

This optimal policy achieved an average return of -109.6 over 10,000 episodes, with maximum and minimum returns of -85.0 and -164.0, respectively. Fig. 13 illustrates the position trajectories for these episodes, demonstrating that the agent successfully reached the goal even under the most unfavorable initial random state. To climb the right hill, the agent plans to use the left hill to develop momentum.

B. Q-learning algorithm

Parametric study

In Q-learning, the number of episodes must be sufficiently large for the agent to visit each state adequately and build an accurate Q-table. In the Blackjack problem that only has only 290 states, we demonstrated that 200,000 episodes were necessary for the learner to converge to a satisfactory optimal policy (see Fig. 7). In contrast, a continuous problem with a finely discretized state space can require millions of episodes to ensure each state is visited a statistically sufficient number of times. However, the Mountain Car operates on a one-dimensional track and state transitions are deterministic, the. This may simplify exploration, potentially reducing the number of episodes needed for convergence compared to more complex environments with an equivalent number of states.

To keep computational costs manageable, we limited Q-learning to 200,000 episodes and tested it on coarser discretization grids of 100×100 and 50×50. As shown in Fig. 14, the 50×50 grid produced a higher expected return (over 10,000 episodes). This is because the agent can explore the smaller number of states in the 50×50 grid more effectively compared to the 100×100 grid. However, the more unstable performance in the 50×50 case may result from coarse bins, which can

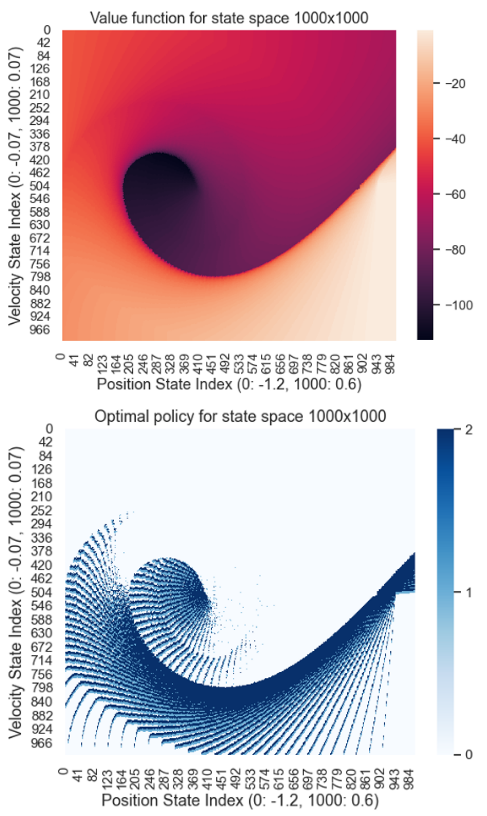

Fig. 12: Heatmaps of the state values and optimal policy generated using the VI and PI algorithms for the Mountain Car problem. The policy discrete values 0, 1, and 2 represent the actions: accelerate to the left (0), don't accelerate (1), and accelerate to the right (2).

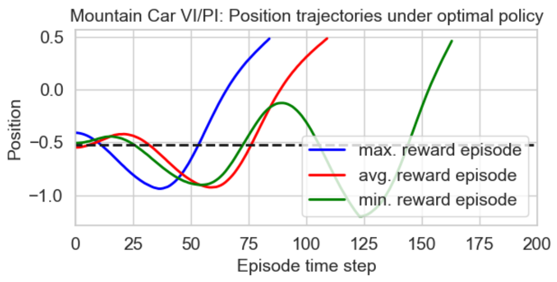

Fig. 13: Car position trajectories yielding the average, max., and min. rewards over 10,000 episodes under the optimal policy of VI and PI using the 1000×1000 state grid.

make state transitions less smooth. For these experiments, the Q-learning parameters were set to: init_alpha=0.5, epsilon_decay_ratio=0.9, min_epsilon=0.10 facilitating epsilon-greedy exploration. The discount factor gamma was set to 0.99.

Fig. 15 shows the expected reward history for the 50×50 state space when using parameters that emphasize exploitation over exploration. In the first scenario, a lower learning rate was applied (init_alpha=0.1) while maintaining epsilon_decay_ratio=0.9 and min_epsilon=0.1. In the second scenario, epsilon decay parameters were reduced (epsilon_decay_ratio=0.1, min_epsilon=0.01) with init_alpha fixed at 0.5. In both cases, the expected reward showed some decrease compared to the original settings (init_alpha=0.5, epsilon_decay_ratio=0.9, min_epsilon=0.1), demonstrating the benefit of exploration for effective learning. On the other hand, it should be noted that reducing the epsilon exploration produces a more stable response, as observed in the Blackjack problem.

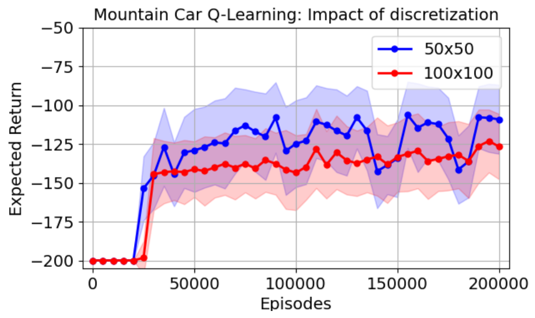

Fig. 14: Impact of the state space discretization on the expected returns Mountain Car using Q-learning.The following parameters were used: init_alpha=0.5, epsilon_decay_ratio=0.9, min_epsilon=0.10. The expected returns were calculated at an interval of 5,000 learning episodes.

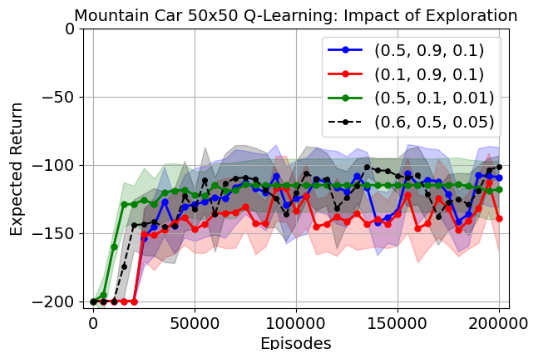

Fig. 15: Impact of exploration ability in Mountain Car using Q-learning and 50x50 state gird. The legend shows the values of the parameter tuple (init_alpha, epsilon_decay_ratio, min_epsilon) for each case.

Optimal policy

Based on the results of Fig. 14, the 50×50 state space was chosen for further tuning of the Q-learning algorithm. To this end, a grid search was conducted covering 32 parameter sets. The search space included init_alpha of 0.5 and 0.6, epsilon_decay_ratio ranging from 0.3 to 0.9, and min_epsilon ranging from 0.01 to 0.2. With the number of episodes fixed at 200,000, the highest expected total return was -101.2 and the corresponding parameter values were: init_alpha=0.6, epsilon_decay_ratio=0.5, min_epsilon=0.05. The corresponding expected return history is shown Fig. 15. The position trajectories yielding the average, maximum, and minimum reward over 10,000 episodes are shown Fig. 16. The minimum and maximum returns recorded over the episodes run were -132.0 and -85.0 respectively.

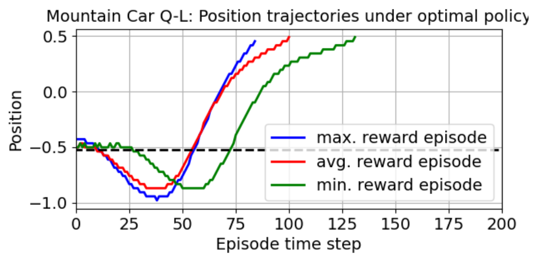

Fig. 16: Mountain car position trajectories yielding the average, maximum, and minimum reward over 10,000 episodes under the optimal policy learned by the tuned Q-learning algorithm.

Interestingly, the tuned Q-learning algorithm outperformed the optimal policy derived from the VI and PI algorithms on the much finer 1000×1000 state grid (which achieved an expected return of -109.6, see Fig. 11 and Fig. 13). This superior performance is likely due to Q-learning's epsilon-greedy exploration, which allows the agent to effectively navigate the 50×50 state space, uncovering a more efficient policy than applying the planning algorithms to the finer discretization. In terms of computational efficiency, Q-learning required 4813.0 seconds for 200,000 episodes. While this is longer than VI on the 1000×1000 grid (396.2 seconds), it is significantly faster than PI, which took 7302.9 seconds. Therefore, Q-learning proves to be a more efficient approach than PI for the Mountain Car problem, even when the transition model is known.

Fig. 17 shows the state values and optimal policy heatmaps produced by tuned Q-learning algorithm. While the lower resolution of the 50×50 grid results in a more coarse representation, the maps still capture the strong patterns seen in the refined heatmap of VI/PI seen in Fig.12. Increasing the number of Q-learning episodes and the discretization level would likely refine these patterns further.

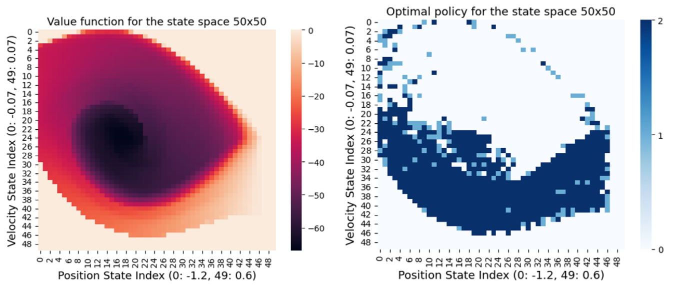

Fig. 17: Heatmaps of the state values (left) and optimal policy (right) generated using with the tuned Q-learning algorithm for the Mountain Car problem.

VI. Conclusions

In this study, we applied the VI, PI, and Q-learning algorithms to the Blackjack and Mountain Car MDP problems. For the Blackjack problem, we anticipated that VI and PI would converge to the same optimal policy, and this was confirmed by our results. It was particularly validating to observe that the generated policy matched the established Basic Strategy rules. Additionally, we expected Q-learning to approximate this optimal policy without surpassing it, which was also confirmed by the results. However, we did not foresee that Q-learning would require at least 200,000 episodes to achieve satisfactory performance. The large number of episode was likely needed due the stochastic nature of the Blackjack game even if it only has 290 states.

For the Mountain Car problem, both VI and PI converged to the same optimal policy across all examined state space refinements, consistent with our expectations. Our experiments revealed that PI imposes a stricter convergence criterion directly on the policy, and for one experiment VI required a significant reduction in the state value tolerance to match PI's results. Unexpectedly, Q-learning outperformed VI and PI, even with a coarser state space discretization. This superior performance was likely due to Q-learning's epsilon-greedy exploration, which allowed it to navigate the state space more effectively and discover a higher-reward policy. We anticipate that increasing the discretization refinement in VI and PI could enhance their policies further, eventually surpassing that of Q-learning. As expected, our experiments also highlighted the importance of carefully tuning Q-learning parameters, including the learning rate and the balance between exploration and exploitation in the epsilon-greedy strategy, to achieve optimal results.

In terms of computational cost, PI proved to be more expensive than VI, as expected, due to the additional iterations required for policy evaluation in each step, alongside policy improvement. For larger state spaces in the Mountain Car problem when the discount factor approached 1, PI became prohibitively costly and impractical for use. In contrast, Q-learning's computational cost scales linearly with the number of episodes, making its total cost more predictable for a given number of episodes.

VII. References

[1] Towers, M. et al. (2024). Gymnasium: A standard interface for reinforcement learning environments. arXiv preprint arXiv:2407.17032.

[2] Barto, A. G. (2021). Reinforcement Learning: An Introduction. By Richard's Sutton. SIAM Rev, 6(2), 423.

[3] Mansfield, J. "bettermdptools" (Version 0.7.2) [Source code]. Available: https://github.com/jlm429/bettermdptools.

[4] Halbersma, R. "gym-blackjack-v1" (Version 1) [Source code]. Available: https://github.com/rhalbersma/gym-blackjack-v1.

[5] Wikipedia. "Blackjack". Available: https://en.wikipedia.org/wiki/Blackjack.