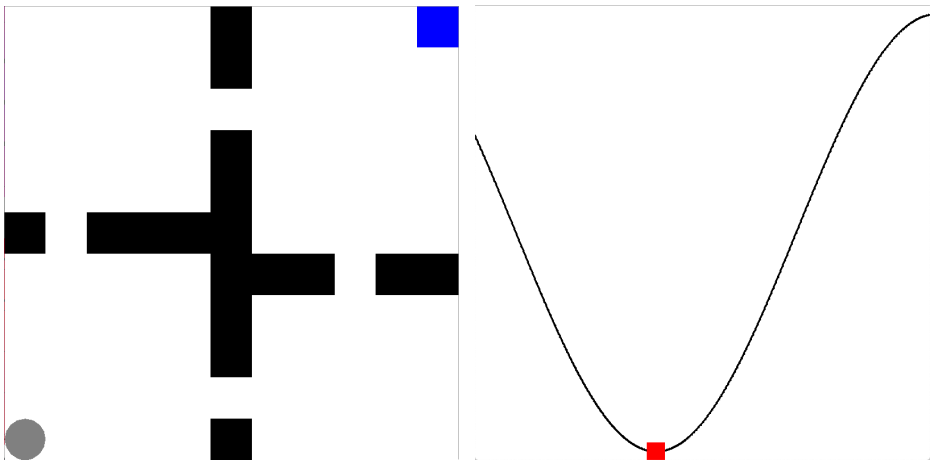
Markov Decision Processes

Peter Lucia

November 24, 2019

1 Abstract

In this work, I explore and analyze the effectiveness of techniques such as Policy Iteration, Value Iteration, and Q-Learning in solving two Markov Decision Processes (MDPs). The first problem is a grid-world based problem where an agent must navigate across four rooms separated by walls to an end goal position. The second problem is a continuous domain problem where a mountain car sitting in a valley must leverage the potential energy of each side of the valley to propel itself forward over a hill. I expect value iteration to show the fastest convergence against the small grid world problem because the inherently smaller state space of the problem makes for a simpler model, and the algorithm is designed to require fewer iterations [11]. I expect Q-Learning to show the faster convergence against the mountain car problem because of the prohibitively large state space would require more iterations for model-based approaches without significant discretiziation.



(a) Four Rooms Problem (b) Mountain Car Problem

Figure 1: Visualization of the MDPs

2 Markov Decision Processes (MDPs)

The four rooms grid world problem is taken from [13] and is implemented with the Brown-UMBC Reinforcement Learning and Planning (BURLAP) java code library. The four rooms grid world problem contains a single agent that must navigate through four corridors to reach a goal position. The grid world consists of 104 states where each room is a 5x5 grid with an additional 4 blocks for each doorway connecting the four rooms. The starting point for the agent is in the bottom left block of the first room, while the goal point is in the diagonal room in the top right-most block. While relatively simple, it is an interesting problem because there are multiple valid shortest paths to the goal and the there is a uniform cost on every block except for the goal block. The shortest diagonal euclidean distance from the starting point to the goal position is blocked by horizontal and vertical walls, which forces the agent to take less optimal paths while prioritizing the long term reward. I expect that model based approaches such as policy iteration and value iteration will converge faster than Q-Learning for this problem because with uniform cost on every non-goal block, the Q-Learning algorithm will have more difficulty learning directly from interaction from the world.

The mountain car problem is taken from [11] and is also implemented with the BURLAP library. In the mountain car problem, the car must propel itself left, right, or coast in order to reach the goal state at the top of the right hill. Unlike the four rooms grid world problem, the mountain car problem is a continuous domain problem with infinite state space. In order to discretize the state space so that the algorithms could operate on the world,

it was necessary to manually sample the position and velocity of the mountain car and construct a policy based on the cross product of the state features and the three possible actions: move left, right, or coast. For example, least squares policy iteration (LSPI) is used to perform action selection without a model within the policy iteration framework [14]. The mountain car problem is interesting because it has a much larger state space than the grid world problem and because it is a more difficult problem for the learning algorithm that I expect will highlight additional advantages and disadvantages of a model based vs. model-free approach.

3 Experiments

3.1 Policy Iteration

One way of finding an optimal policy in a finite MDP is by obtaining a sequence of monotonically improving policies and value functions, such that there is a guarantee of improvement between successive operations. The process, called *policy iteration*, is guaranteed to converge due to the limited number of policies possible for a finite MDP [11]. The policy iteration algorithm is defined in [11] as follows:

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Figure 2: Policy Iteration Algorithm

To define convergence criteria, against the four rooms problem the policy iteration algorithm was set to terminate either when the maximum change in the value function became smaller than 0.001 or when the number of iterations reached 1000. Against the mountain car problem, the policy iteration algorithm was set to terminate after either a change of less than 10^{-6} occurred between iterations or until 30 iterations total had passed, as recommended by the BURLAP library authors [13].

Figure 3 shows the policy heat map created by the PI algorithm against the four rooms problem. To achieve this policy, the discounting factor, γ was set to 0.99, which will cause the agent to overwhelmingly prefer future reward. Against the four rooms problem, the Policy Iteration algorithm converged after only 4 iterations. Against the mountain car problem, the policy iteration algorithm converged after only 6 iterations. Without the linear function approximation or discretization method, which sampled transition tuples from the trajectory of the car, the policy iteration algorithm did not converge due to the infinite state space of the continuous MDP. A policy heat map of the state space for the continuous MDP is not shown because the BURLAP library does not provide an API for LSPI state space visualization. It is recommended that future experiments develop potential methods for visualization of a sampled state space from a continuous MDP that can be integrated into the existing BURLAP library.

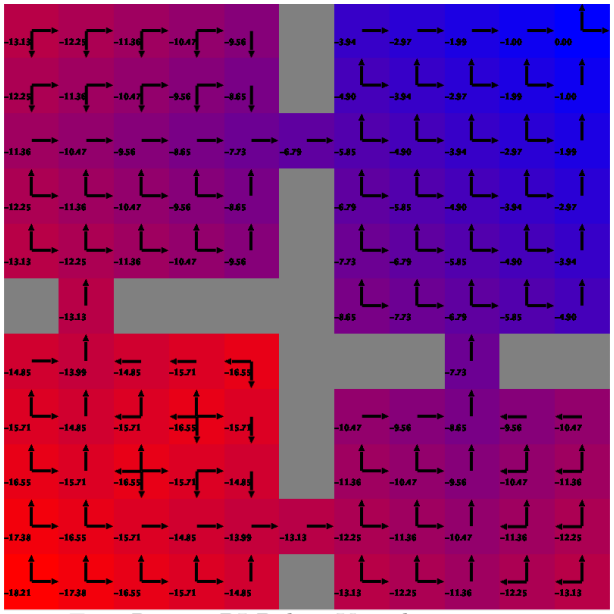
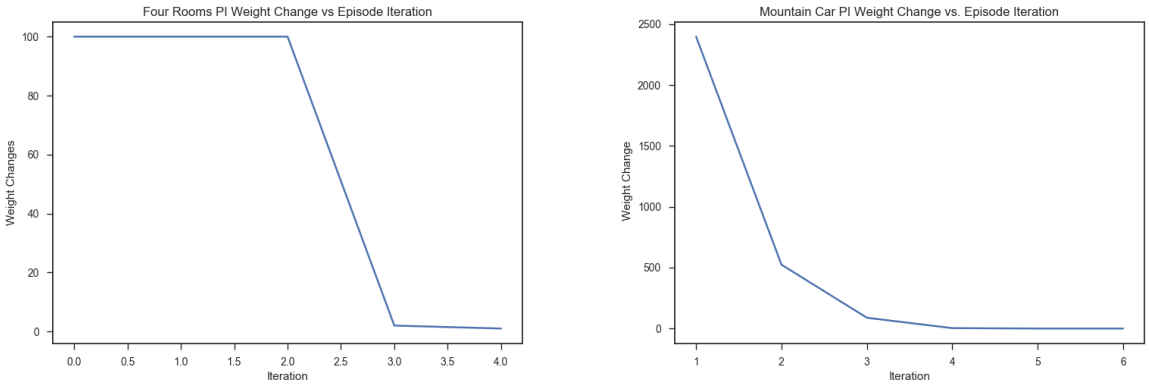


Figure 3: Four Rooms PI Policy Visualization at 6 iterations

Figure 4 shows the magnitude of policy iteration weight changes between each successive iteration in both the Four Rooms and Mountain Car problems. In the initial iteration, the weights are set to infinity. It took the policy iteration algorithm 6 iterations to converge to where the weights did not shift more than 10^{-6} from the previous iteration against the mountain car problem and 4 iterations to converge against the four rooms problem. The four rooms problem is simpler, and the weight changes were never greater than 100. As would be expected with a larger state space and complexity inherent in the the mountain car problem, the weight changes were ten orders of magnitude greater, initially at 2500 before settling close to 0 after the 6th iteration.



(a) Four Rooms (b) Mountain Car

Figure 4: Policy Iteration Weight Change vs Episode Iteration

Figure 6 shows the effect of changing the discount factor, γ on the total reward per episode against the four rooms problem. A score of -999 indicates that the policy iteration algorithm did not find the goal position within the 1000 iterations, which was set as the maximum number of possible iterations before the algorithm would give up. A score of -20 indicates that the algorithm reached the goal position, as there are 20 steps between the starting position and the goal position in the four rooms problem, with each non-goal cell accruing a cost of -1. There is a drastic, positive effect on total reward when the discount factor moves from 0.52 to 0.53. This suggests that the policy iteration algorithm must value future rewards slightly more than current reward in order to converge against the four rooms problem.

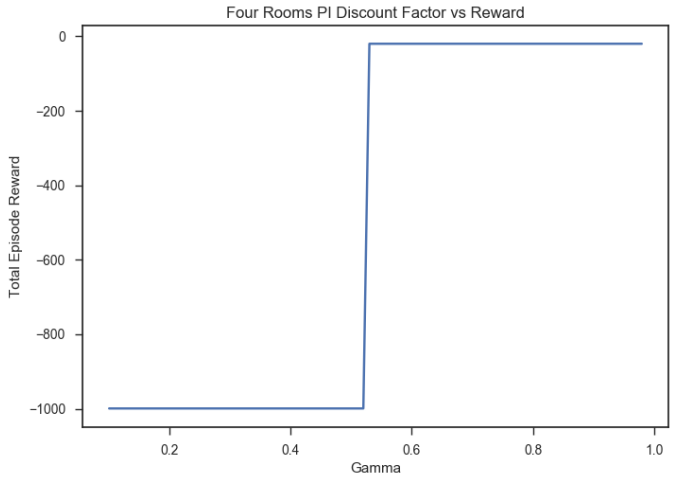


Figure 5: Four Rooms Policy Iteration Discount Factor vs. Reward

3.2 Value Iteration

Value Iteration is a model-based approach that is almost identical to Policy Iteration, but differs in that its update rule requires the maximum to be taken over all actions. Value iteration typically offers faster convergence because the evaluation is stopped after a single sweep over each state, while policy iteration requires multiple sweeps through the state set [11]. The value iteration algorithm is defined in [11] as follows:

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $\quad v \leftarrow V(s)$
| $\quad V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$
| $\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

Figure 6: Value Iteration Algorithm

Given that the value iteration algorithm is almost identical to policy iteration, but is designed to offer faster convergence, we would expect the two algorithms to converge to the same policy for simpler grid world problems, but in different amounts of time. Figures 8 and 3 show the algorithms did converge to the same policies against the four rooms problem. Figures 5 and 11 suggest instead that the policy iteration algorithm took less time per episode than value iteration. Figure 14 shows the cumulative and average rewards for the Q-Learning algorithm against the four rooms problem. The Q-Learning algorithm took 50 episodes to consistently obtain a positive average reward per episode, but required only 1.62 ms per episode, about 1/5 as much time as policy iteration, and 1/100 as much time as the value iteration algorithm, as shown in Figure 11. To define convergence criteria, the value iteration algorithm was set to terminate either when the maximum change in the value function became smaller than 0.001 or when the number of iterations reached 100. With a discount rate of 0.99, against the four rooms problem, the value iteration algorithm converged in many cases after 20 iterations, requiring 15 more iterations than the Policy Iteration algorithm against the same problem. This was an unexpected result as the value iteration algorithm is expected to take fewer iterations than the policy iteration algorithm because it requires the maximum action be taken over all actions available during each sweep of the state space [11]. However, these results suggest that the max operation used by value iteration, which injects a policy improvement step into the policy evaluation step, may sometimes hinder time to convergence. One possible explanation for this is that the uniform cost reward function used by the BURLAP library in the four rooms problem may take away the benefits normally offered by the max operation of the value iteration algorithm. In other words, if every possible action in every position offers the same reward except for when the agent is adjacent to the goal position, the value iteration algorithm would offer little

benefit. It is recommended that future experiments investigate other grid worlds with non-uniform cost functions to further explore this result.

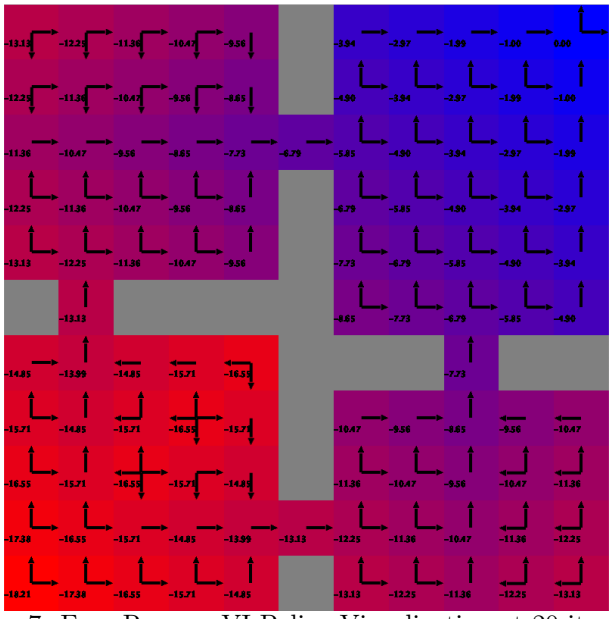
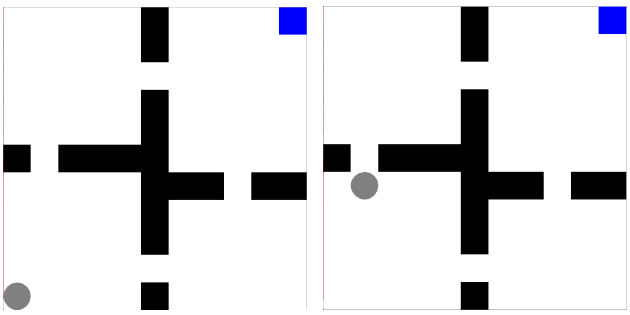


Figure 7: Four Rooms: VI Policy Visualization at 20 iterations

Figure 9 shows the progression of the agent toward the goal position using the optimal policy determined by the value iteration algorithm. There are two viable directions that agent can select from to reach the goal position with the fewest number of steps when starting from the bottom left position. It is recommended that future experiments move the starting position around the grid to see if there are any tradeoffs with any of the algorithms.



(a) VI: Four Rooms Start (b) VI: Four Rooms Mid 1

(c) VI: Four Rooms Mid 3 (d) VI: Four Rooms Finish

Figure 8: Four Rooms: VI Progress Visualization

Figure 5 shows a comparison of the episode duration for policy iteration vs. value iteration. Interestingly, value iteration took longer overall, and required 20 episodes to converge to a duration of under 2ms. There was less

variation with policy iteration, as the average episode duration remained under 2 ms for all 50 episodes.

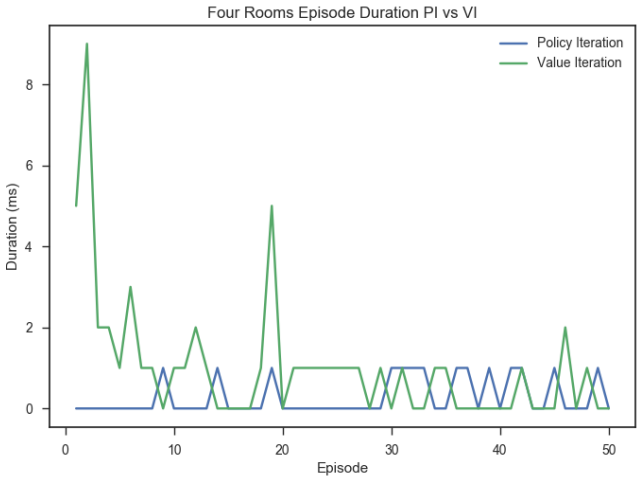


Figure 9: Four Rooms Episode Duration Policy Iteration vs. Value Iteration

Figure 10 shows the effect of the discount factor, γ, on the total episode reward for the four rooms problem. For $\gamma > 0.67$ the value iteration algorithm reached the goal state with only 20 steps. Otherwise, reaching the goal state took more steps or did not happen within the max iteration limit of 1000 steps.

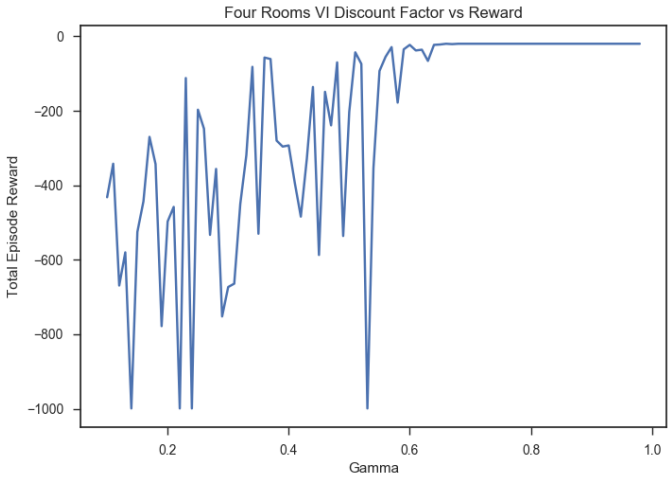


Figure 10: Four Rooms Value Iteration Discount Factor vs Reward

TestName	Duration (ms)
Four Rooms VI	223.00
Four Rooms PI	5.00
Four Rooms Q-Learning	1.62

Figure 11: Four Rooms Avg. Episode Duration

3.3 Q-Learning

Q-Learning is one of the most effective model-free algorithms for learning from delayed reinforcement. This is in part because the Q values will converge regardless of how the agent behaves during data collection [9]. The Q-Learning rule is defined as:

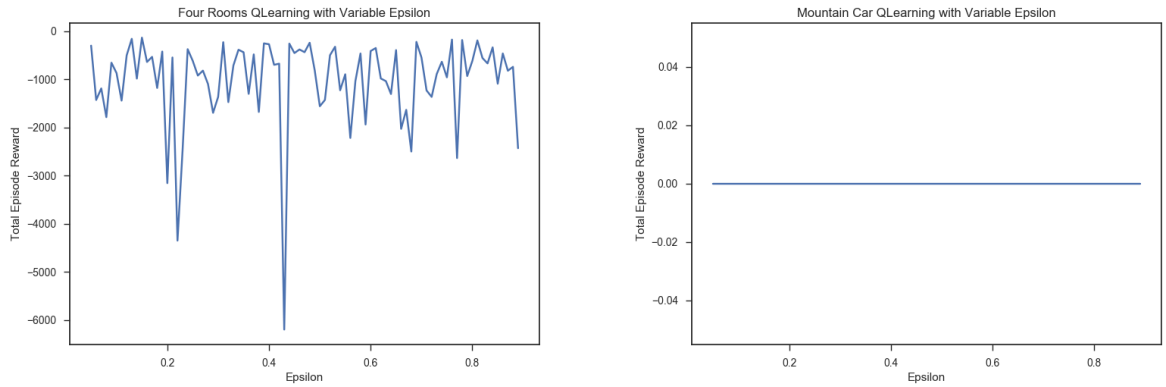
$$Q(s,a) \coloneqq Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

where the tuple $< s, a, s', a' >$ contains the experience information of the current and future state. The quantity $Q(s, a)$ contains the Q values for the current state. The quantity $r + \gamma \max_{a'} Q(s', a') - Q(s, a)$ emcompasses the discounted future reward, with $\max_{a'} Q(s', a')$ containing the maximum possible future reward in the new state s'. The probability that the algorithm converges is dependent on whether α, the learning rate, is set to decay appropriately. The discount factor, γ, is set to between 0 and 1. When set close to 0, current rewards are given much higher importance, while if set close to 1, the agent will strive to acheive longer-term, delayed reward [9, 10]. The QLearning algorithm defined in [11] is as follows:

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 12: QLearning Algorithm



(a) Four Rooms (b) Mountain Car

Figure 13: Exploitation vs. Exploration: Varying Epsilon

One of the challenges with reinforcement learning strategies such as Q-Learning is the tradeoff between exploitation and exploration [11]. When epsilon is set lower, the agent must exploit what it has learned in order to achieve reward. On the other hand, if it is set higher, the agent prioritizes exploration in order to gain delayed reward [11]. For non-epsilon exploratory Q-Learning experiments, the default epsilon value of 0.1 was used. Otherwise, epsilon was varied from 0.10 to 0.99. Figure 13 shows the results from varying epsilon against both the four rooms and mountain car problems. There was no clear trend to using a higher or lower value of epsilon. One possible explanation for this is because of the uniform negative reward at all coordinate positions except for the goal position in the four rooms problem. Additionally, a uniform negative reward is given in the mountain car problem for all locations in the valley except the goal position. In future experiments, it may be interesting to investigate the effect of placing increasing rewards in x positions leading up to the goal position at the top of the hill. While the Q-Learning algorithm did converge against this problem, it required over 40 more iterations to do so than the policy iteration algorithm. However, in this experiment, the Q-learning algorithm is interacting directly with the continuous domain, while the policy iteration algorithm is acting in a sampled, discretized domain. Therefore, it is possible that the faster convergence was instead because the policy iteration algorithm was operating on a sampled domain. It is recommended that future experiments against the mountain problem compare the performance of model-based and model-free approaches against a single, discrete domain.

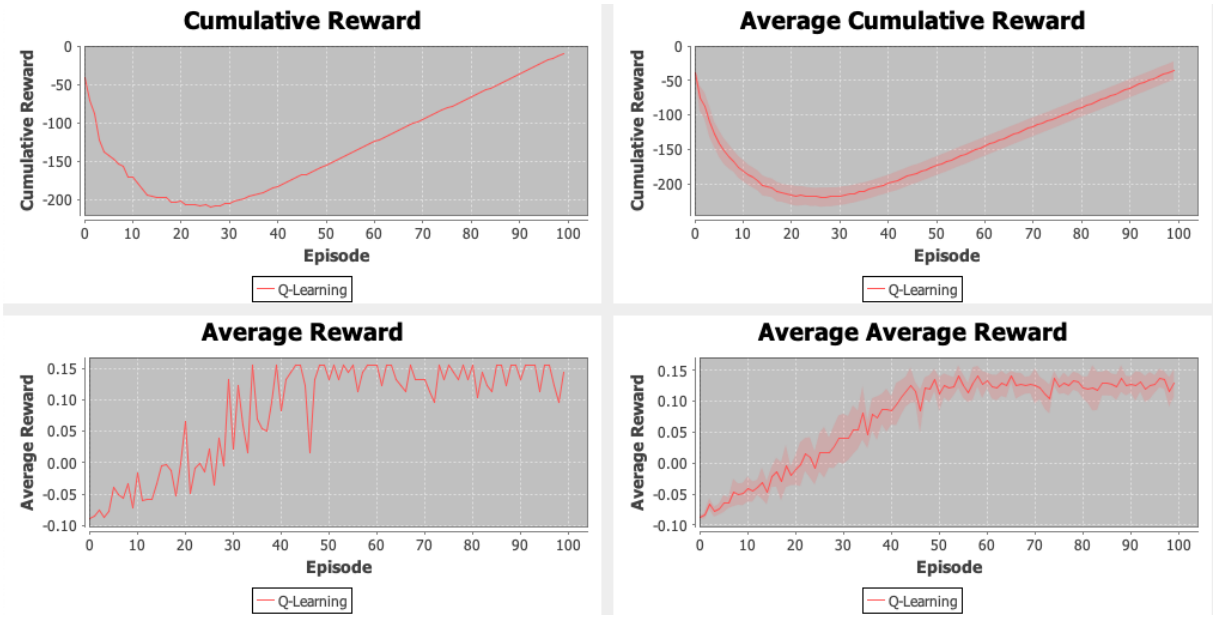


Figure 14: Four Rooms: QLearning Results

Figures 14 and 15 show the Q-Learning rewards for each episode. The average reward converged after 50 episodes to approximately 0.15 against the four rooms grid problem. Against the mountain car problem, the average reward did not increase significantly with each episode, and ranged between 0 and 0.0025.

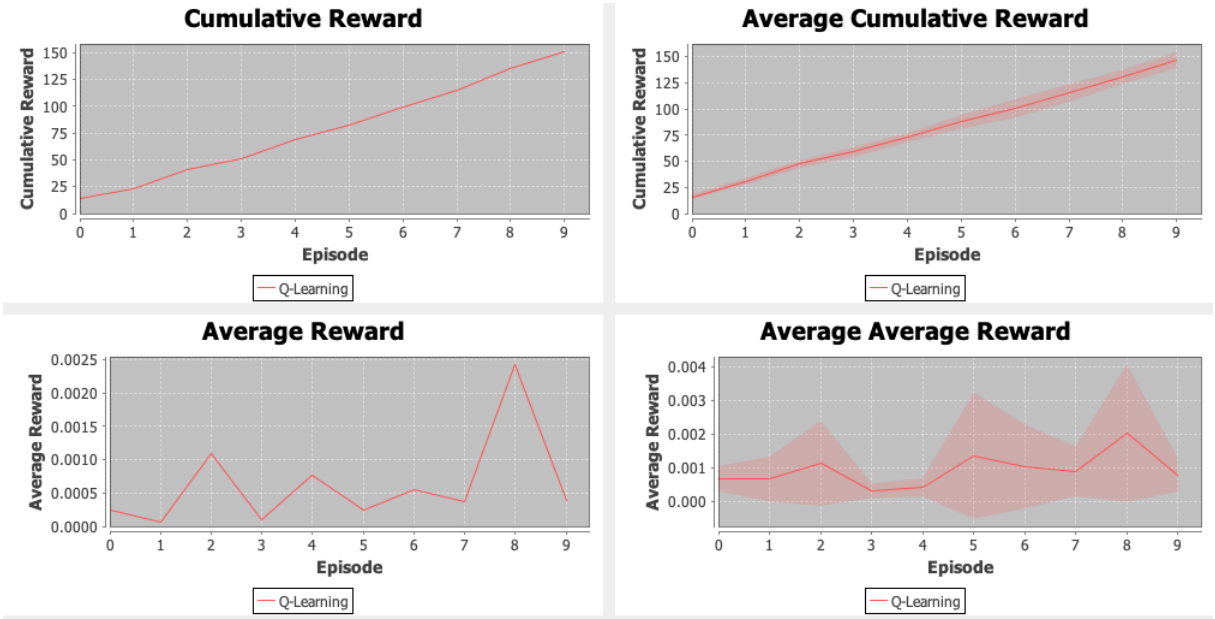


Figure 15: Mountain Car: QLearning Results

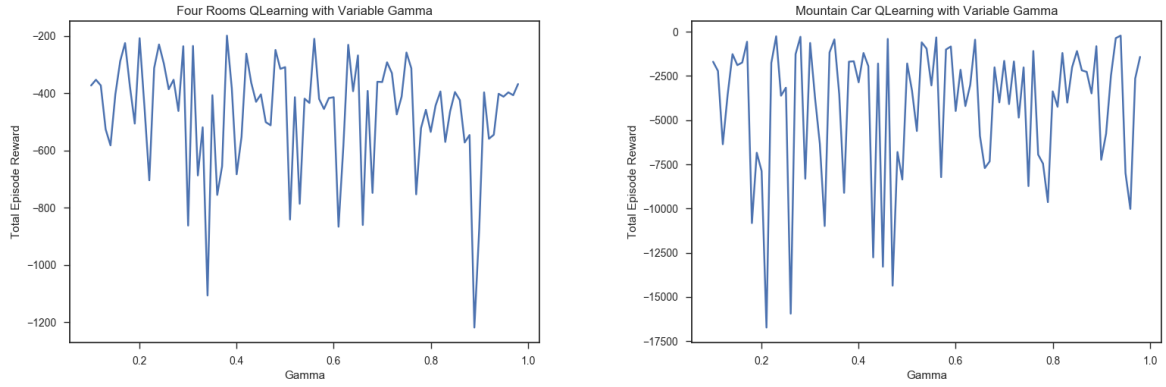
Figures 11 and 16 show the average episode duration for each algorithm. Interestingly, the Q-Learning algorithm had the lowest duration per episode across both problems, taking an order of magnitude less time when compared with the policy iteration algorithm against the four rooms and mountain car problems.

TestName	Duration (ms)
Mountain Car PI	3415.00
Mountain Car Q-Learning	161.42

Figure 16: Mountain Car Avg. Episode Duration

Figures 17a and 17b show the effect of varying the discount factor, γ, from 0.10 to 0.99. Interestingly, against the

four rooms problem, these result suggest that there was no clear benefit for a lower or higher gamma. Conversely, against the mountain car problem, the total episode reward shows a decrease in variance and a slight upward trend with increasing values for gamma, suggesting that the agent would fare better if prioritizing long term rewards over the short term.



(a) Four Rooms (b) Mountain Car

Figure 17: Varying the Discount Rate (Gamma)

4 Conclusion

Contrary to expectation, the policy iteration algorithm required less time per episode than the value iteration algorithm for the four rooms problem, while the Q-Learning algorithm required the lowest average time per episode against both the four rooms and mountain car problems. Despite requiring different amounts of time per episode and for convergence, the value iteration and policy iteration algorithms converged to the same policy against the four rooms problem. Also, if the discount rate was set below 0.53 for policy iteration, the algorithm never converged, but for larger values it would consistently converge. For the value iteration algorithm, the discount rate needed to be above 0.67 for the algorithm to converge consistently. The Q-Learning algorithm demonstrated optimal per-episode speed against the mountain car problem, requiring 161 ms per episode. With a monotonically increasing discount rate, the variance in the total episode reward was reduced for the Q-Learning algorithm, but interestingly against the four rooms problem, the discount rate had no clear effect on total episode reward. In general, it is recommended that future experiments investigate the effect of a non-uniform reward cost for both problems, as this may have taken away the benefits normally offered by the max operation of the value iteration algorithm. One challenge of this experiment was to discretize the state space of the continuous domain mountain car problem into a finite set of states that the policy and value iterations could build policies for. This is one advantage to the Q-Learning approach, which interacts directly with the world and is model free. It is recommended that future experiments develop methods for visualization of a sampled state space from the mountain car problem, as such a visualization would be more revealing of the model-based approach's policies against continuous domain problems.

References

[1] Mitchell, Tom M. "Machine learning." (1997).

[2] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of machine learning research 12.Oct (2011): 2825-2830.

[3] Buitinck, Lars, et al. "API design for machine learning software: experiences from the scikit-learn project." arXiv preprint arXiv:1309.0238 (2013).

[4] https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

[5] https://archive.ics.uci.edu/ml/datasets/Wine

[6] Garris, Michael D., James L. Blue, and Gerald T. Candela. "NIST form-based handprint recognition system." Technical Report NISTIR 5469 and CD-ROM, National Institute of Standards and Technology. 1994.

[7] https://classroom.udacity.com/courses/ud262

[8] http://www.scikit-yb.org/en/latest/

[9] Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." Journal of artificial intelligence research 4 (1996): 237-285.

[10] https://en.wikipedia.org/wiki/Q-learning

[11] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[12] Alpaydin, Ethem. Introduction to machine learning. MIT press, 2014.

[13] http://burlap.cs.brown.edu/index.html

[14] Lagoudakis, Michail G., and Ronald Parr. "Least-squares policy iteration." Journal of machine learning research 4.Dec (2003): 1107-1149. APA