CS 7641 Machine Learning
Robert Bobkoskie

Assignment 4:
Markov Decision Processes

1. INTRODUCTION

The objective of this project is to explore Markov Decision Processes (MDPs), and their use in value/policy iteration and Q-learning applications. MDPs offer a method to formalize non-deterministic events, which are events where the actions and outcomes are uncertain. I chose to experiment with MDPs using a gridworld, which is an agent confined to a maze (grid). Non determinism was modeled by assigning a probability, P(A) = 0.82, that the agent takes a specified action (A), and (1 - P(A) / (Σ(num actions) - A) that an agent does not take A. The probability of not taking the specified action (1 - P(A) is known as 'noise'. In my gridworlds, the noise is 1.00 - 0.82 = 0.18, and was equally divided among the set of actions, less A.

2. PROBLEMS

2.1. mdptoolbox.mdp: Build Gridworld, Transition and Rewards Functions

For this assignment, my intent was to create two sets of MDPs based on gridworld. The first set would be the standard two-dimensional gridworld, with a set of actions: [North, South, East, West], a transition function (T), T[s, a, s'], where P(s' | s, a), and a reward function R(s, a, s'). The second set of problems would expand the 2-d gridworld to a 3-d gridworld, allowing for additional movement in the z direction: [North, South, East, West, Z1, Z2], where Z1, Z2 are movement in opposite directions along the Z dimension. Note that for both sets of MDPs, [T, R] are matrices of a form that could be input into `mdptoolbox.mdp`[1], which is a python library that provides classes and functions for the resolution of MDPs.

The first step was to construct a gridworld matrix. Numpy vectorization and slicing were used to generate gridworld matrices of two or three dimensions, populating each cell with the appropriate reward (terminal, living or NaN). The gridworld matrix was then input into functions that created both T and R as the agent 'acted' and moved about the gridworld.

For each of 2-d, 3-d gridworld, T was created based on the conditional probability of entering a state (s'), given the current state (s) and an action P(s' | s, a). For example, given a gridworld with: actions (A), P(A) = 0.82, and P(noise) = 0.18, T can be constructed by considering the probabilities of an agent either exiting or remaining in a state. An agent would remain in a state if the action were to take the agent out of bounds, the agent attempted to enter an inaccessible state (modeled using NaN values), or the agent ends the game from a terminal state. An agent would exit s (and enter s') based on the non-deterministic model, e.g., if the action is North, the agent will go North 82% of the time and, provided the actions are valid, will go either: South, East, or West 0.06% of the time. Consider the case that some actions from a state are invalid for an agent. For example, in figure 1a, if an agent is in state [0, 2], the action North and South would not be a valid, however, East and West are valid. So, if North or South are the intended actions, the agent can go East or West with a probability of 0.06 for each. Thus, the agent can leave s [0, 2] with P = 0.12, and will remain in s with P = 1 - 0.12 = 0.88.

The R matrix was constructed by gather rewards located in each cell in gridworld. There were relatively large rewards located in terminal (game ending) states, and a smaller (positive) living reward (penalty when negative) in all other accessible cells. The NaN values in the gridworld are used to designate inaccessible states.

The framework for generating transition and reward matrices was flexible enough that T_m and R_m could be generated from a gridworld of any size or shape. For this project, 2-d gridworlds of sizes: [4x4, 8x8, 16x16, 32x32], and 3-d gridworlds: [4x4x4, 8x8x8, 16x16] were selected. Two terminal states with large positive and negative rewards are located to the right of the inaccessible cells of both the 2-d and 3-d gridworlds. NaN values were located in four cells in the center of the grid, designated them as inaccessible. All other cells had a small living reward of ±0.01.

Figure 1a: A 2-d, 4x4 gridworld.

```
[[ -0.01  -0.01  -0.01  -0.01]
 [ -0.01    nan    nan  10.  ]
 [ -0.01    nan    nan -10.  ]
 [ -0.01  -0.01  -0.01  -0.01]]
```

Figure 1b: The transition (T_m) matrix created from the gridworld shown in figure 1a.

```
[[[ 0.88  0.06  0.   ...,  0.    0.    0.  ]
  [ 0.06  0.88  0.06 ...,  0.    0.    0.  ]
  [ 0.    0.06  0.88 ...,  0.    0.    0.  ]
  ...,
  [ 0.    0.    0.   ...,  0.88  0.06  0.  ]
  [ 0.    0.    0.   ...,  0.06  0.88  0.06]
  [ 0.    0.    0.   ...,  0.    0.06  0.12]]

 [[ 0.12  0.06  0.   ...,  0.    0.    0.  ]
  [ 0.06  0.88  0.06 ...,  0.    0.    0.  ]
  [ 0.    0.06  0.88 ...,  0.    0.    0.  ]
  ...,
  [ 0.    0.    0.   ...,  0.88  0.06  0.  ]
  [ 0.    0.    0.   ...,  0.06  0.88  0.06]
  [ 0.    0.    0.   ...,  0.    0.06  0.88]]

 [[ 0.12  0.82  0.   ...,  0.    0.    0.  ]
  [ 0.06  0.12  0.82 ...,  0.    0.    0.  ]
  [ 0.    0.06  0.12 ...,  0.    0.    0.  ]
  ...,
  [ 0.    0.    0.   ...,  0.12  0.82  0.  ]
  [ 0.    0.    0.   ...,  0.06  0.12  0.82]
  [ 0.    0.    0.   ...,  0.    0.06  0.88]]

 [[ 0.88  0.06  0.   ...,  0.    0.    0.  ]
  [ 0.82  0.12  0.06 ...,  0.    0.    0.  ]
  [ 0.    0.82  0.12 ...,  0.    0.    0.  ]
  ...,
  [ 0.    0.    0.   ...,  0.12  0.06  0.  ]
  [ 0.    0.    0.   ...,  0.82  0.12  0.06]
  [ 0.    0.    0.   ...,  0.    0.82  0.12]]]
```

Figure 1c: The rewards (R_m) matrix created from the gridworld shown in figure 1a.

```
[[ -0.01  -0.01  -0.01  -0.01]
 [  0.     0.    -0.01  -0.01]
 [  0.     0.    -0.01  -0.01]
 [  0.    10.    -0.01  -0.01]
 [ -0.01  -0.01   0.     0.  ]
 [  0.     0.     0.     0.  ]
 [  0.     0.     0.     0.  ]
 [  0.    10.     0.     0.  ]
 [ -0.01  -0.01   0.     0.  ]
 [  0.     0.     0.     0.  ]
 [  0.     0.     0.     0.  ]
 [-10.     0.     0.     0.  ]
 [ -0.01  -0.01  -0.01  -0.01]
 [  0.     0.    -0.01  -0.01]
 [  0.     0.    -0.01  -0.01]
 [-10.     0.    -0.01  -0.01]]
```

As will be discussed in the results section of this report, I was disappointed with the results obtained from mdptoolbox.mdp. The results for policy/value iteration, however, I was concerned

about how mdptoolbox.mdp implements Q-learning. In particular, the T and R functions are assumed, but unknown to a Q-learner. An optimal policy (∏*) is identified through a combination of exploration and exploitation. However, the mdptoolbox.mdp.QLearning function takes T and R as parameters, which seems to contradict Q-learning. So, after spending ~80 hours developing methods to generate MDP transition and rewards matrices for any 2-d or 3-d gridworld, I decided to investigate other tools to explore policy/value iteration and Q-learning.

2.2. Berkeley AI Tool-Kit

Fortunately, I found a toolkit[2] from a Berkeley AI course that required only slight modifications to demonstrate MDP solutions via policy, value, and Q-learning. Although not as flexible in allowing for analysis of iterations and convergence, other options allowed for better analysis of the affect the discount function gamma (γ) and living reward has on ∏*: policy/value iteration and Q-learning. Two gridworld MDPs were selected, each having actions [N, S, E, W]. The shape of the two gridworlds differed slightly: 3x4, 3x7. but the big difference was in the assignment of rewards and arrangement of the terminal states, and the modeling of non-deterministic actions.

One gridworld (3x4) had two terminal states with values ±1 in the two upper rightmost states. Non-deterministic movement was modeled by assigning P(s' | s, a) = 0.8 for a = [N], and 0.1 for [E, W]. The second gridworld, known as 'bridgeworld' (3x7) assigned very large negative rewards (-100) to terminal states running along the entire top and bottom rows, thus creating a 'bridge' of states down the center. At the end of the bridge was a terminal state with a much smaller reward of +10. Non-deterministic movement was modeled by assigning P(s' | s, a) = P(A), and P(¬A) for the 'noise'. The interplay of γ, the 'noise' and the living reward provided interesting results, particularly for bridgeworld.

3. DISCUSS THE RESULTS

The objective of this project was to evaluate sets of gridworld MDPs of varying sizes for both 2-dimensional, and 3-dimensional configurations using mdptoolbox.mdp. Some concerns with how mdptoolbox.mdp implements Q-learning led me to investigate other MDP tool-kits to better evaluate policy/value iteration and Q-learning. I was able to gather useful information for the number of iterations, and performance numbers using mdptoolbox.mdp, and these results will be discussed. Additional work using the Berkeley AI Tool-Kit will expand the interplay between the MDP problem solving methods, reinforcement learning (Q-learning), as well as the parameters (γ: *Discount*, *Noise*, *Living Reward*, for policy/value iteration, and α: *Learning Rate*, ε: *Action Selection* for Q-Learning) available for tuning these algorithms (Bellman equations).

3.1. Value and Policy Iteration

Given an MDP with transition, T[s, a, s'], or P(s' | s, a) and rewards, R(s, a, s') functions, value and policy iteration can be used to solve them. The solution for an MDP is an optimal policy, ∏*(s) which is the optimal action from state (s) that generates an optimal utility (the expected utility starting in (s) and acting optimally).

Policy iteration is a two-step process, with evaluation and improvement functions. The evaluation step, $V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')]$ assumes a fixed policy (∏), thus eliminates evaluating max values over all actions for a given state. The evaluation step runs at $O(S^2)$. The improvement step, $\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi_i}(s')\right]$ runs like value iteration $O(S^2A)$. Policy iteration updates the utilities (with a fixed policy) over several iterations, thus the optimal utility function is a consequence of identifying ∏*.

Value iteration, $V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$ considers all actions, and updates both the value and (implicitly) the policy at every iteration. Value iteration runs at $O(S^2A)$, which is costly if the number of actions is large. Policy is not implicit from value iteration, but can be extracted from values: $\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$. Policy extraction from Q-values is trivial: $\pi^*(s) = \arg\max_a Q^*(s,a)$.

3.1.1. mdptoolbox.mdp

The results in this section will demonstrate the number of iterations and length of time to solve ever more complex MDPs. The experiments involved 2-d gridworlds of sizes: [4x4, 8x8, 16x16, 32x32], and 3-d gridworlds: [4x4x4, 8x8x8, 16x16]. Convergence was not achieved, e.g., the program ran for hours without returning for 2-d gridworlds with shape 2^6x2^6 or greater, and 3-d gridworlds with shape 2^5x2^5 or greater.

Experimental results showed that both value and policy iteration solved MDPs, both methods returned the same ∏* and utilities. However, Q-learning did not return a ∏* that matched policy or value iteration. This could be due to limitations on how mdptoolbox.mdp implements Q-learning. For instance, there was no parameters to adjust (α: *Learning Rate*, ε: *Action Selection*). These limitations might not allow for enough exploitation. Only the results for a 4x4 2-d gridworld are shown, as the results from other 2-d and 3-d gridworlds were similar.

```
For A = {'N':0, 'S':1, 'E':2, 'W':3}:

∏* (POLICY): (2L, 2L, 2L, 1L, 2L, 0L, 0L, 1L, 2L, 0L, 0L, 1L, 0L, 0L, 0L, 1L)

VALUES: (0.03193170495999999, 0.32034938303999994, 2.0532643763200005, 12.35187358272, -
0.00015054528000000008, 0.0, 0.0, 12.48, -0.00015054528000000008, 0.0, 0.0, 0.0, -
0.0102717664, -0.00014856384000000008, -1.9814400000000013e-06, -1.7280000000000016e-08)

∏* (Q-POLICY): (1L, 0L, 2L, 1L, 3L, 0L, 0L, 1L, 3L, 0L, 0L, 1L, 1L, 0L, 1L, 1L)
```

Experimental results were inconsistent in identifying the most efficient (run-time, figure 3) method for solving MDPs. My concern is that for large (32x32) 2-d gridworlds, policy iteration took far more time to converge than Q-learning. Recall that an agent running Q-learning has no knowledge of T, R. Therefore, it would be expected that some exploration/exploitation by the agent would be required to arrive at a solution. In theory, this should take more time, I question how mdptoolbox.mdp implements Q-learning. When comparing the 2-d and 3-d results in figures

2 and 3, it's important to consider the maximum size of the grid under evaluation. For example, although 3-d gridworld adds a Z dimension, the maximum size (16x16) is exponentially smaller than the max gridworld in 2-d (32x32). So, at first glance, it may appear that the number of iterations and run-time are out of balance, this can be explained by the exponentially larger 2-d gridworld. The interesting observation is that value iteration requires a constant number of iterations, independent of gridworld size or dimension, while policy iteration requires fewer iterations (figure 2), and less time (figure 3) to converge for smaller gridworlds. Recall that policy iteration has evaluation and improvement steps. In general, policy iteration spends more time evaluating $O(S^2)$, and less time improving $O(S^2A)$. For gridworlds with a smaller state space, this fives policy iteration an advantage. With a larger state space, value iteration performs better in terms of number of iterations and run-time to convergence. This can be explained by the small number of actions, 4 for 2-d and 6 for 3-d, as well as acyclic MDPs (no looping between states). Under these conditions, it was expected that value iteration would perform better. Policy iteration would be expected to perform better when there is a large number of actions for any size gridworld.

Figure 3b: Run-time for convergence for Policy iteration for 3-d gridworlds of increasing sizes.

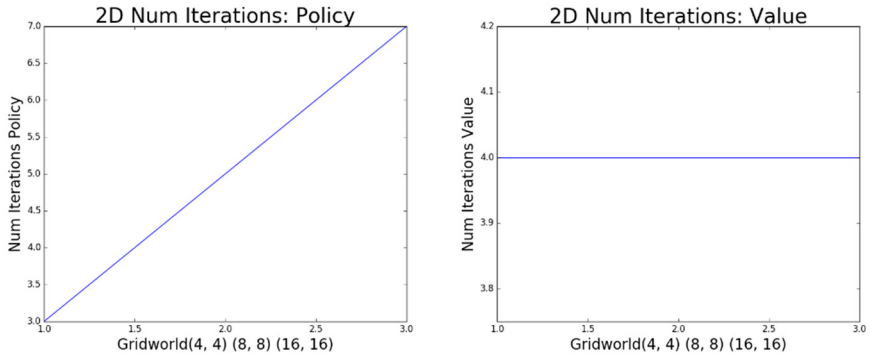


Figure 2a: Number of iterations for convergence for Policy iteration for 2-d gridworlds of increasing sizes.

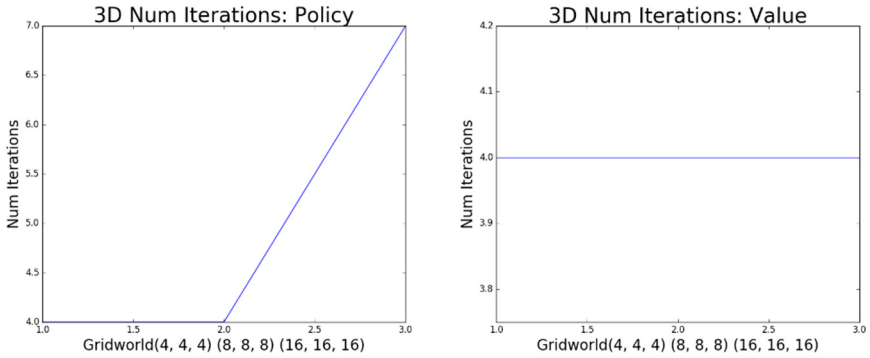


Figure 2b: Number of iterations for convergence for Policy iteration for 3-d gridworlds of increasing sizes.

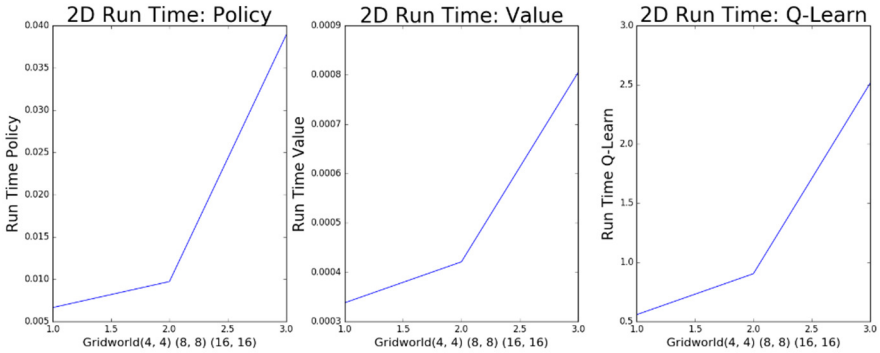
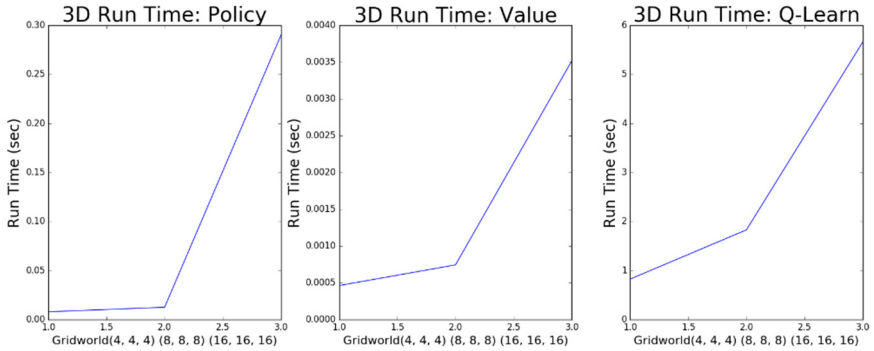


Figure 3a: Run-time for convergence for Policy iteration for 2-d gridworlds of increasing sizes.



3.1.2. Berkeley AI Tool-Kit

The Berkeley AI tool-kit (Berkley-TK) was used to further explore the relationship between the MDP problem solving methods, and the parameters (γ: *Discount*, *Noise*, *Living Reward*) available for tuning these methods.

The results in figure 2 show the affect the living reward and discount, γ has on ∏* (depicted by the arrows in the states). Small γ not only discounts future rewards, but also influences ∏* (compare the lower right state in figures 3a, 3c). Adjusting the living reward ever so slightly (figures 3a, 3b) has an even greater impact on ∏*.

Figure 3a: A gridworld MDP with the following parameters: γ=0.9, living reward=0.0, terminal rewards=[1.0, -1.0].

Figure 3b: A gridworld MDP with the following parameters: γ=0.9, living reward=0.11, terminal rewards=[1.0, -1.0].

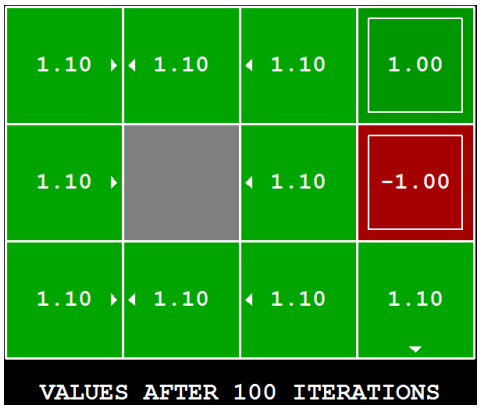
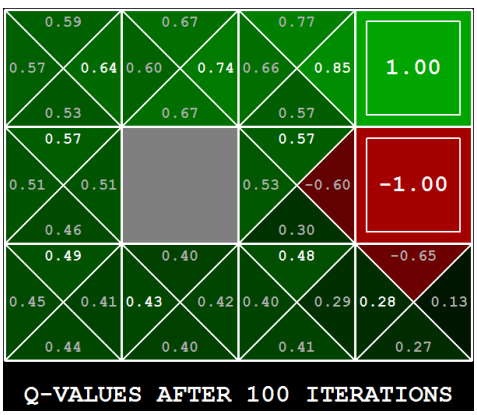
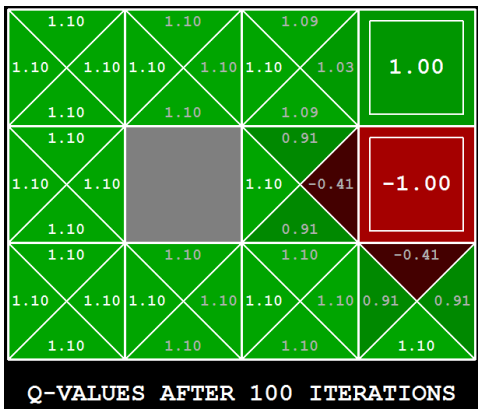
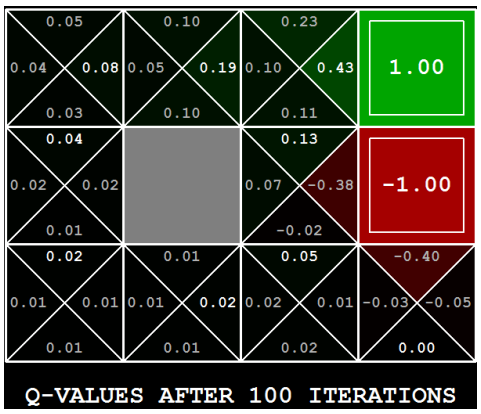


Figure 3c: A gridworld MDP with the following parameters: γ=0.5, living reward=0.0, terminal rewards=[1.0, -1.0].







Q-values are also show in figure 3. They show all ∏ and utilities. Q-values are used for Q-learning, which will be discussed further in later sections of this report.

A second MDP, bridgeworld was evaluated with the Berkeley-TK. This MDP had very large terminal penalties along the top and bottom rows, with a much smaller terminal reward at the end of the middle row. The objective is to adjust the living reward, noise (uncertainty of moving East) or the discount of future rewards, γ. As shown in figure 4, a combination of low uncertainty, and a large discount on future rewards was enough to get the agent find a ∏* to cross the bridge (figure 4b). This can be explained by the large discount on future rewards, which was large enough to discount the large negative utilities at the perimeter of the bridgeworld. The low uncertainty eliminated 'doubt' in the actions, and allowed the agent to act on the discounted future rewards. The results in figure 4c show that if the living reward and discount on future rewards are achieved, the same ∏* to cross the bridge can be achieved. However, the optimal utilities are different, as expected.

Figure 4a: A bridgeworld MDP with the following parameters: γ=0.001, living reward=0.0, noise (uncertainty)=0.2, terminal rewards as shown.

Figure 4b: A bridgeworld MDP with the following parameters: γ=0.9, living reward=0.0, noise (uncertainty)= 0.0001, terminal rewards as shown.

Figure 4c: A bridgeworld MDP with the following parameters: γ=0.9, living reward=20.0, noise (uncertainty)=0.2, terminal rewards as shown.

3.2. Reinforcement Learning: Q-learning

Unlike value and policy iteration, reinforcement learning solves an 'unknown' MDP. More specifically, the transition and reward functions are not known. An agent explores an environment, receiving rewards and percepts (states, s'). The objective is to learn ∏* by maximize the rewards over T:

- Explore: Try unknown actions to get information.
- Exploit: Use what is gathered from exploration.

- Regret: Taking sub-optimal actions during exploration, making mistakes.

There are two distinct approaches to reinforcement learning, model based, and model free learning. In model based learning, based on experience, an approximate model (MDP) is constructed. It is assumed the model is correct. The agent obtains values by counting s' for each (s, a), and then normalizes to give an estimate of T[s, a, s']. R[s ,a, s'] is discovered as the agent experiences (s, a, s'). Essentially, the agent is reconstructing the MDP and performing value iteration.

For this project, a model free learning method, Q-learning was implemented to examine reinforcement learning. More precisely, Q-learning is a temporal difference learner, which is a prediction-based machine learning method. Q-learning assumes the existence of an MDP, a state space, a set of actions, transition (T[s, a, s']) and reward (R(s, a, s')) functions. With no knowledge of T or R, the objective is to identify the optimal policy (∏*) by evaluating optimal actions (Q-values), and computing all averages over T using sampled outcomes. Q-learning is implemented by sampling Q(s, a): $Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$, and incorporating this sample into a running average of Q(s, a): $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$. Here, the alpha parameter is analogous to a learning rate for the agent. A larger alpha places more weight on new samples, which implies faster, less stable learning because more weight is given to each sample. Using Q-values, Q(s, a) instead of values, V(s) at state (s) allows Q-learning to converge to the optimal policy (off policy learning), even if the agent does not act optimally. This implies that if the agent explores enough, and the learning rate is made small enough, then action selection does not matter. Rather than have the agent explore for a long time, possible indefinitely, action selection can govern exploration, thus enabling more exploitation. One approach is an epsilon greedy policy, where an agent will act randomly (explore) with some (small) probability (ε), otherwise, act according to the current policy (exploit). For this project, an epsilon greedy method was implemented. Other approaches, such as exploration functions better optimize the balance between exploration and exploitation. One such function that was not implemented, takes a value estimate, u and a visit count, n, and returns an optimistic utility: $f(u,n) = u + k/n$. The Q update rule, $Q(s,a) \leftarrow_{\alpha} R(s,a,s') + \gamma \max_{a'} Q(s',a')$ can be modified by replacing the Q-value with this 'boosted' Q-value: $Q(s,a) \leftarrow_{\alpha} R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$. Using this value estimate for action selection would enable an agent to transition quickly to exploitation after exploration, thus eliminating many random sub-optimal actions in an extended exploration phase. It is expected that this value based exploration function would require fewer actions to converge to ∏*, thus minimize regret (the difference between the rewards obtained during Q-learning and the R* achieved if ∏* were known and followed) over the epsilon greedy approach.

The Berkley-TK was also used to evaluate an epsilon greedy policy (ε) for adjusting exploration. For this brigeworld MDP, more exploration (figure 5), e.g., a higher value for ε, performs better with fewer episodes (iterations). As the number of episodes is increased (5a, 5b, 5c), a lower (but not too low, 5d) value for ε leads to better results (5d, 5e, 5f), where better results are defined as a more complete ∏*. As expected, a greater the number of episodes leads to a better ∏* (5g).

However, picking the right value for ε is a challenge. I expect that an exploration function that utilizes a 'boosted' Q-value would better make adjustments between exploration and exploitation, and thus achieve better performance in 'learning' over the epsilon greedy policy. At a minimum, I would expect that an exploration function would take some of the ad-hoc approach toward choosing a 'good' value for ε. As shown in this small sample set of experiments, the value for ε has a large impact on ∏*, making the choice of an optimal value for ε hard to identify.

Figure 5a: A bridgeworld Q-learning with ε=0.001. Time to complete 100 episodes: 32.46 sec.

Figure 5b: A bridgeworld Q-learning with ε=10. Time to complete 100 episodes: 33.28 sec.

Figure 5c: A bridgeworld Q-learning with ε= 1000. Time to complete 100 episodes: 34.58 sec.

Figure 5d: A bridgeworld Q-learning with ε= 0.1. Time to complete 500 episodes: 148.17 sec.

Figure 5e: A bridgeworld Q-learning with ε= 1. Time to complete 500 episodes: 167.38 sec.

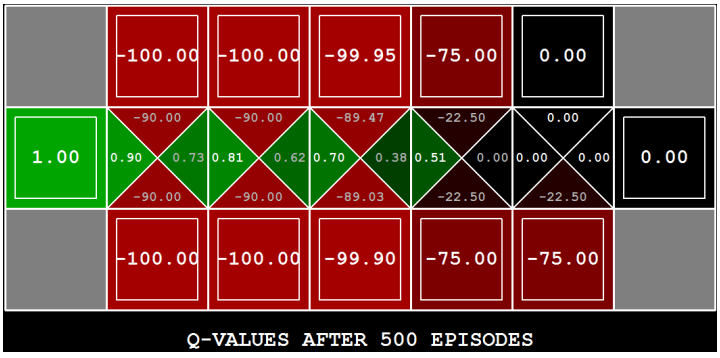
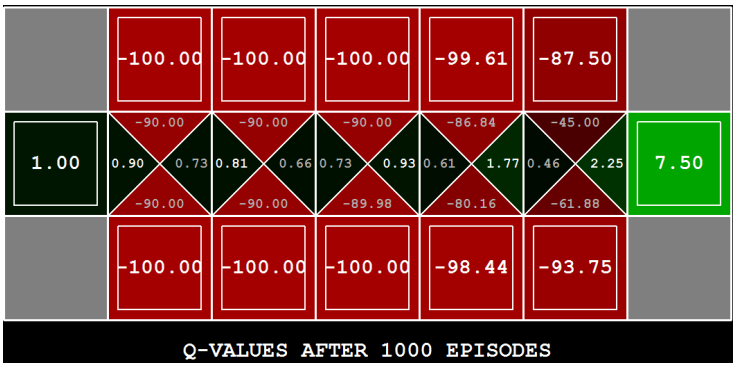


Figure 5f: A bridgeworld Q-learning with ε= 10. Time to complete 500 episodes: 167.38 sec.

Figure 5g: A bridgeworld Q-learning with ε= 1.
Time to complete 1000 episodes: 427.51 sec.



4. CONCLUSION

I started this project with great ambitions to evaluate a unique set of gridworld MDPs, including a 3-dimensional configuration using mdptoolbox.mdp. This didn't work out exactly as anticipated, and I had to shift to another framework (Berkeley AI Toolkit) to evaluate MDPs. This setback gave me a chance to further explore the interplay between the MDP problem solving methods, reinforcement learning (Q-learning), as well as the parameters (γ: *Discount*, *Noise*, *Living Reward*, for policy/value iteration, and α: *Learning Rate*, ε: *Action Selection* for Q-Learning) available for tuning these algorithms (Bellman equations). Everything was not lost using mdptoolbox.mdp, as I was able to discover the relationship between the complexity of an MDP and performance (number of iterations and run-time). One last note, the equations shown in this report were cut from various lecture notes provided by the Berkeley AI class cited in (2). The formatting is strange, but it was more efficient than entering the equations using MS Word.

---

[1] http://pymdptoolbox.readthedocs.io/en/latest/api/mdp.html#module-mdptoolbox.mdp
[2] https://inst.eecs.berkeley.edu/~cs188/sp12/projects/reinforcement/reinforcement.html