Markov Decision Process

Problems

A common problem for agents rooted in a real world inspired situation like many video games or even agents in the real world is how to navigate their surroundings and over come the obstacles they meet. A simplified view of this problem will be explored in this paper. In this paper, the problem known as Gridworld is explored. A single agent can move around in a 2d grid with the goal of reaching some goal coordinate. Obstacles such as walls are added to make the problem more difficult. The movements in the world are non-deterministic. Each movement has X chance to succeed. The remaining 1-X is split evenly among the other directions. Unless specified differently, X is held to be 80%. This paper explores two maps. The first a simple 10x10 space setup as the classic four room map. As seen below.

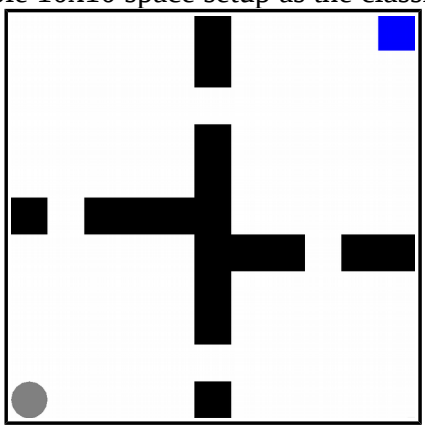

Figure 1 Small Map

The second is a much larger problem with a 25x25 map with much more walls. Additionally, it has two goal coordinates as seen below.

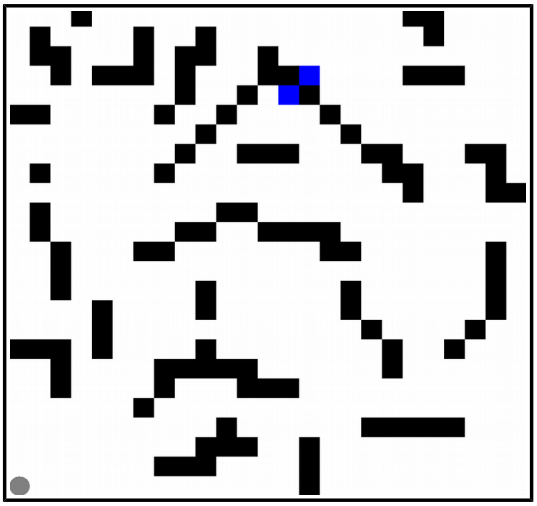

Figure 2 Large Map

The difference between the two will create interesting differences in the problems that need to overcome. The second map has over a 500% increase in the number of states. Additionally, the addition goal state will change the behavior as nearby states could favor moving in different directions towards different goal states. Additionally, it can easily seen how to apply this sort of problems to actual problems such as Roombas who operate on similar levels on complexity.

Value Iteration

In Value iteration, a value for each state is calculated. This value is the maximum reward that an agent could achieve in k steps starting at the given state. This is run until convergence.

The method for the value iteration experiments was as follows. Probability of success, discount, and reward values were each iterated over in turn. Probability of success was otherwise held to 80%. Discount was otherwise held to 0.99. Reward was otherwise held to 5. This experiments were run to completion and their clock time was record. It is important to note some clock times are excessively larger than their iteration count would suggest this maybe due to some anomalies in how the CPU handled threads. The results of the experiments are below.

Small

probability	0.99	0.9	0.8	0.6	0.4	0.2
iterations	12	18	26	51	79	242
runtime	730	288	265	438	691	1179
discount	0.99	0.95	0.9	0.8	0.6	n/a
iterations	26	23	20	15	9	n/a
runtime	99	168	145	103	64	n/a
reward	100	10	5	3	1	-0.1
iterations	30	27	26	25	22	28
runtime	110	2901	100	368	145	403

large

probability	0.99	0.9	0.8	0.6	0.4	0.2
iterations	17	30	38	69	136	287
runtime	3261	1134	1655	2353	3873	8721
discount	0.99	0.95	0.9	0.8	0.6	n/a
iterations	38	34	26	20	9	n/a
runtime	1366	1298	838	1030	560	n/a
reward	100	10	5	3	1	-0.1
iterations	43	40	38	38	34	43
runtime	2402	1336	1132	1279	2863	1836

Figure 3: Value Iterations Results

As the success probability increases from 0.2 to 0.99, the required number of iterations to achieve convergence decays rapidly with 0.99 requiring 1/20 of the iterations. Thinking about how value iterations works, it is looking at the value of a state but not directly at the actions while probability of success is directly effecting actions making it much more difficult to create the model necessary to get optimal values. Interestingly, despite its difficulties with low success probabilities it was able to figure out at 0.2 success probability, it is more likely to go in the direction desired by going in any other. Thus often attempt to move the direct opposite of the direction were picked. Additionally, the value of a given state radially increase from the goals spaces and increase in value as success probability increases. This is logical since as success probability goes up, the actions are more likely to succeed thus picking the best set is possible. Interestingly, the small map approaches the large map in iterations as the probability decreases while the runtime stays far, far above. This suggest the extra difficult of the two final locations and massively increased complexity of the map does not provide as much challenge to VI as the changes in probability does; however, each iteration cost a lot more due to the expensive nature of the value update algorithm.

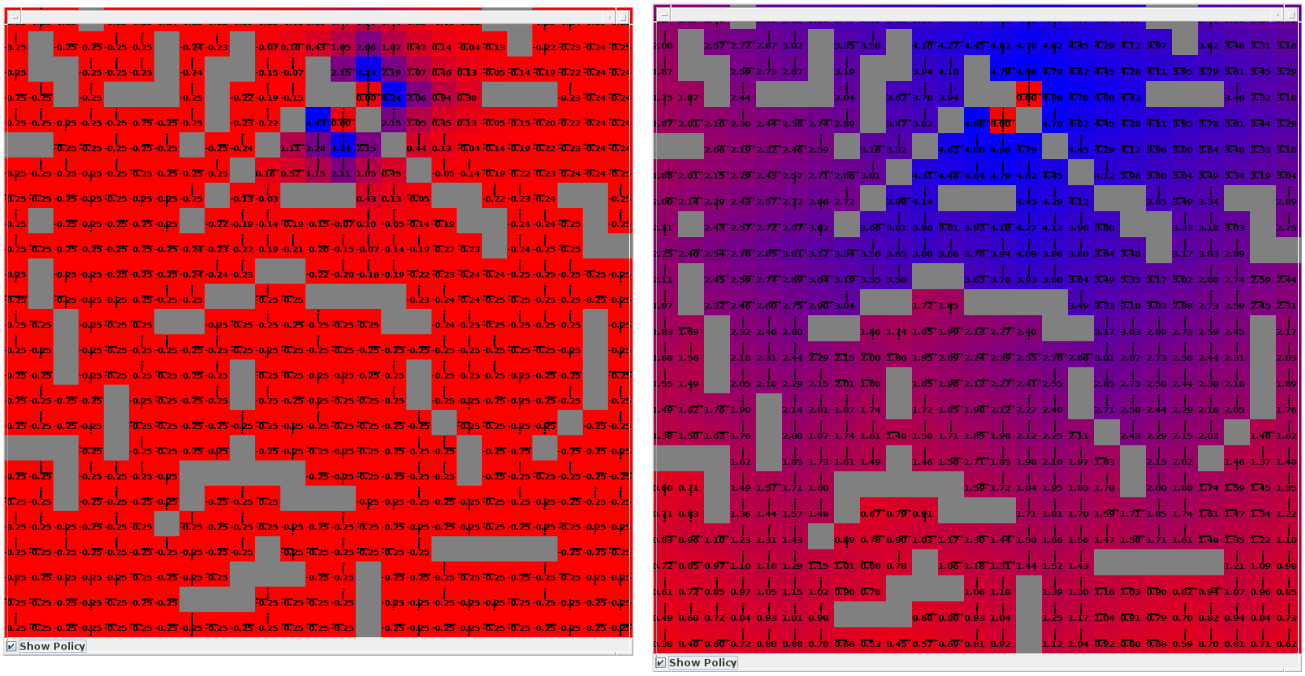

Figure 4 Value Iteration convergence for Success Probabilties 0.2(left) and 0.9 (right)

As the discount increases from 0.6 to 0.99 the iterations grew larger as well. This is interesting as the difficult of solving the problem increases as the rewards are worth more at each state. This suggest both problems have many states which suffer from high immediate awards while the long run award of those path is lower than other options. The low discount corrects this issue allowing the algorithm to find the correct solution more quickly. However, the solution to 0.6 discount is very odd in some states. For instances there is a few cases for the next move being into a wall or moving away from the goal state.

As the goal reward value increases from 1 to 100 the number of iterations increases. This is due to the increasing rewards resulting in a much longer time to reach convergence effectively obfuscating the correct values due to magnitude in changes each iterations. As value decreases the cost of movement can fight the propagated reward value to create a more clear gradient of rewards quicker resulting in faster convergence. Interestingly, decreasing from 1 to -0.1 also increases the number of iterations for both maps. This suggests that a sufficiently negative goal reward can have a similar effect as a large positive reward. Except in this case, instead of initial obfuscating convergence; it instead initially creates a negative value in the direction of the goal causing initial values to not move towards the goal since they are updated based on neighbors and only the nearby states to the goal know to move towards it. The rest see a negative gradient towards the goal at the start.

Policy Iteration

In policy iteration, the approach of the algorithm is very similar; however, instead of calculating the value of a given state, the value of a action within a state is calculated. This means the value of every state is calculated based on the current policy which is iteratively updated until convergence. Policy iteration directly looks at the actions as compared to value iterations indirectly looking at them.

The experiment for policy iteration is the exact same as the value iterations experiments. The iteration and runtime results can be seen below. The first thing to note is how much larger the number of iterations and runtime are for policy iterations over value iterations. Typically by at least an order of magnitude for both numbers. It is important to note that the number of iterations listed in the table are the number of inner iterations. Policy iterations goes through phases of iterating before updating the policy. Limiting the number of iterations could result in a less optimal solution in much shorter time. Another important note is that Policy iteration and value iteration both converge to roughly the same result. The main difference is in a few states have different values and different directions but largely they do not affect how optimal the solution is. These were largely in states far from the goals which only would be entered if started there or nearby.

Small

probability	0.99	0.9	0.8	0.6	0.4	0.2
iterations	697	645	557	344	508	478
runtime	10076	4419	3153	2997	2869	2856
discount	0.99	0.95	0.9	0.8	0.6	n/a
iterations	557	186	117	62	28	n/a
runtime	2719	1303	1099	690	288	n/a
reward	100	10	5	3	1	-0.1
iterations	728	564	557	554	549	598
runtime	3999	2901	2399	3291	2965	3074

large

probability	0.99	0.9	0.8	0.6	0.4	0.2
iterations	1076	913	842	444	714	526
runtime	41452	33111	29864	19686	26192	19622
discount	0.99	0.95	0.9	0.8	0.6	n/a
iterations	842	269	142	65	31	n/a
runtime	27217	12609	8006	3436	2365	n/a
reward	100	10	5	3	1	-0.1
iterations	1068	884	842	822	816	798
runtime	38627	31074	28212	26566	28101	33317

Figure 5 Policy Iteration Results

Varying the probability results in a very different effect on policy iteration than value iteration. The runtime of policy iteration decreases from 0.99 to 0.6. Also, 0.5 and 0.8 were relatively the same runtime. Looking at the split among the individual inner iterations the reason for this becomes more clear. The number of inner iterations decreases with probability suggesting that the nondeterministic movements allow the algorithm to explore more of the space within fewer iterations result in less policies needing to be explored. 0.4 however, the likelihood of moving in any given direction has increases so much that it is favoring exploring over exploiting to the point where it starts to slow down the inner iterations too much resulting in increased runtime. Interestingly, 0.2 resulted in a drop in runtime over 0.4. This suggest once the probability has decreased to the point that choosing a direction is picking the least likely direction to move; the algorithm can compensate and pick actions accordingly, These observations between how to the two algorithms work makes sense since policy iteration is directly looking at the effect of actions.

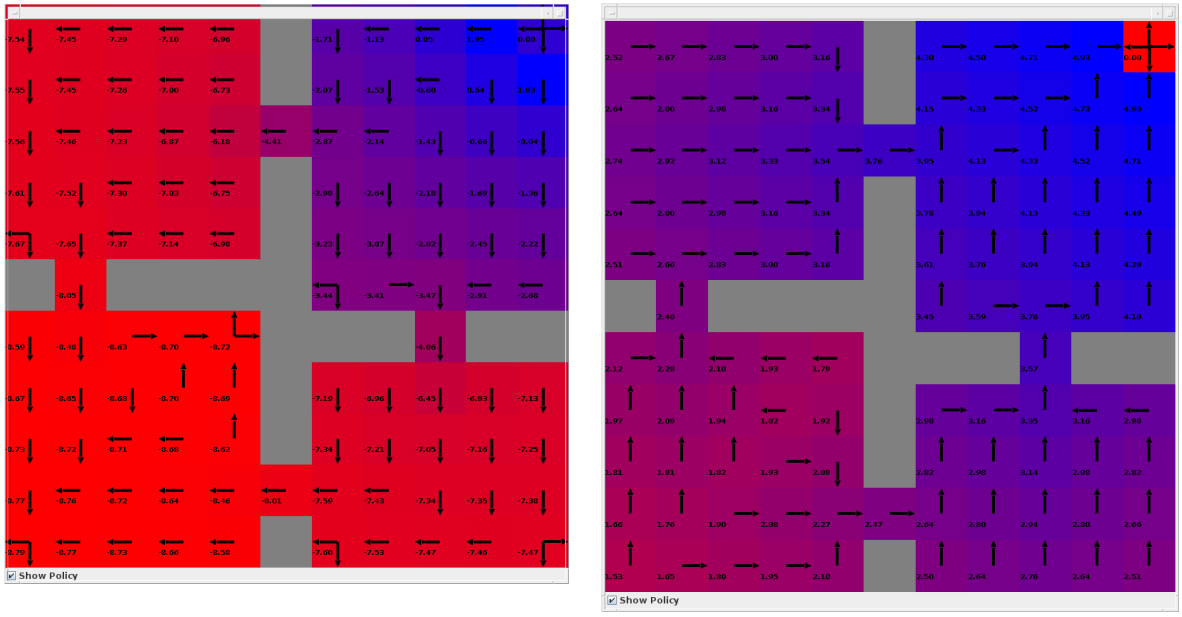

Figure 6 Small Map with 0.2 and 0.9 Success Probability

Varying the discount value had the same effect as in value iteration but much more pronounced. The number of iterations dropped from 842 to 31 for the large map. Interestingly, the small map was nearly the same number of iterations at lower discount values really illustrates how strong the effect is on the large map. However, unlike value iteration, policy iteration was able to find a much better solution without moving into walls or away from the goal at lower discount values. This suggest that policy iteration can handle putting off rewards much better than value iteration.

Varying the reward value has a different effect on the result for the large map but the same effect on the small map. The number of iterations of the large map decreases with rewards including -0.1 while it goes up for the small map and both maps for value iterations. All of the iterations on the large map have same number of policy iterations while the inner iterations change. This suggest that smaller values allow it to do less iterations to see a small enough change to reach convergence while overall since it knows the actions that will tend towards the goal it will be able to find a good policy.

Q-Learning

Q-learning differs from the previous two algorithm significantly. Value and policy iterations know a lot about the MDP including the transition model, the rewards for all states, etc, so they are effective given everything they need to find an optimal policy and have to calculate it from there. Q-learning, however, is given very little and has to learn from its experiences. Q-learners initialize all states to an estimate Q-value which encodes the value for moving into a state. It then can move state to state and receive a reward as it enters a state and use that reward to update the Q-value. As the Q-values are estimate or possibly on some level randomly initialized, the Q-learner needs to explore the state space to get an accurate view. In this paper epsilon greedy exploration is used to determine how much exploration to exploiting is done to assist the learner in exploring the space. Epsilon greedy exploring is similar to simulated annealing where one value encodes how much to explore, Q, and one value encodes how fast to switch from exploring to exploiting, epsilon. This way when it initially knows nothing it explore then transition to exploiting once it learns, hopefully.

The experimental design for Q Learning involves varying the initial Q while holding epsilon to 0.1 and epsilon values while holding the initial Q to be 0.3. The effect on number of steps per episode, the average reward per episode, and the final policy were observed and recorded. The runtime of the experiments can be seen below. Small initial Q and epsilon naturally reduce the time of the Q learner as they determine how long until it begins to stop exploring and try to converge. Interestingly, the smallest values for initial Q start to increase the runtime instead. This suggest that it continues to try to converge for longer before finding local optima within the space.

Small

Q initial	0.3	0.5	1	5	30
runtime	5285	3919	3662	7226	31119
Epsilon	0.05	0.1	0.3	0.5	0.8
runtime	4446	5888	7113	9644	24439

large

Q initial	0.3	0.5	1	5	30
runtime	16502	13948	34532	42660	121593
Epsilon	0.05	0.1	0.3	0.5	0.8
runtime	25830	27799	30389	36137	234385

Figure 7 Q Learning Results

For the variation of the initial Q value, the small map was able to converge to the same rewards and number of steps for all values. However, as Q increases the number of episodes before it begins to trend upwards. At 30 for the initial Q, it does not start trending upwards until around 300 episodes. This makes sense as initial Q forces the learner to explore longer and as the number of states is small it can easily solve the problem. The exception being the super small Q value which slows down exploration to the point where it starts to suffer from not finding nearby better paths for the small map for longer. However, for the larger map, the large Q values were unable to converge to as good of a solution as small Q values, but they are trending upwards suggesting that it may eventually converge to a equivalent solution if allowed more episodes. The larger map problem allows for the learner to discover an optimal path without dropping into a local optimal. Overall for these problems, larger Q values for the same epsilon extend the searching period which slows down the overall solution.

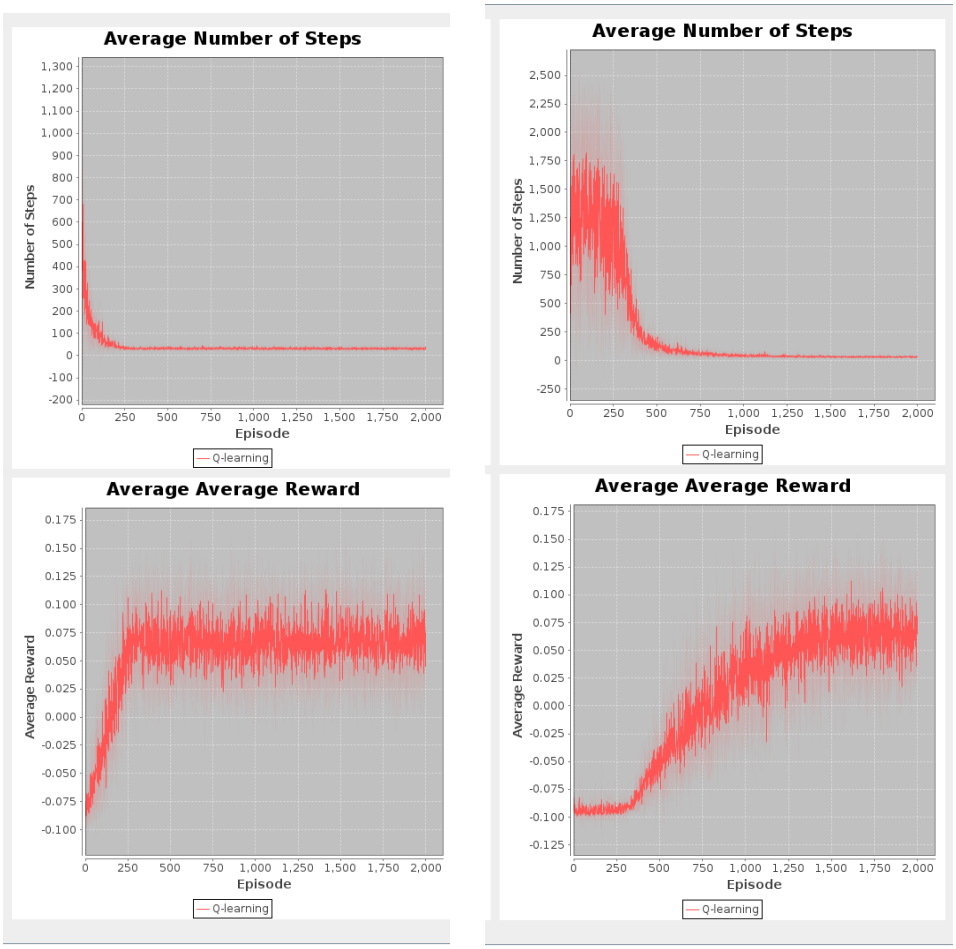

Figure 8 Graph of Q-learner performance with initial Q of 0.5 (left) and 30 (right)

As epsilon increases, the learner transitions from exploration to exploitation faster. In both cases, the smaller epsilon converges to a better solution. For the small map, the 0.05 and 0.1 converge to same average award and number of steps. This suggest that it converged to a optima. However, the larger map always increases in reward while decreasing in number of steps as the epsilon values decreases. This suggest that exploration is very important in the larger map which makes sense since the state space is much larger especially given the small initial Q value. For both small and large maps, they increase in runtime as epsilon increases. Notably, 0.8 is an order of magnitude over all the other values really shows how long the exploration phase was resulting in taking much more steps each episode. However in all cases, the largest epsilon values move towards their final values much quicker reaching the maximum levels at around 250 and the extra episodes and thus runtime were not helpful and could not have been run with the same result.

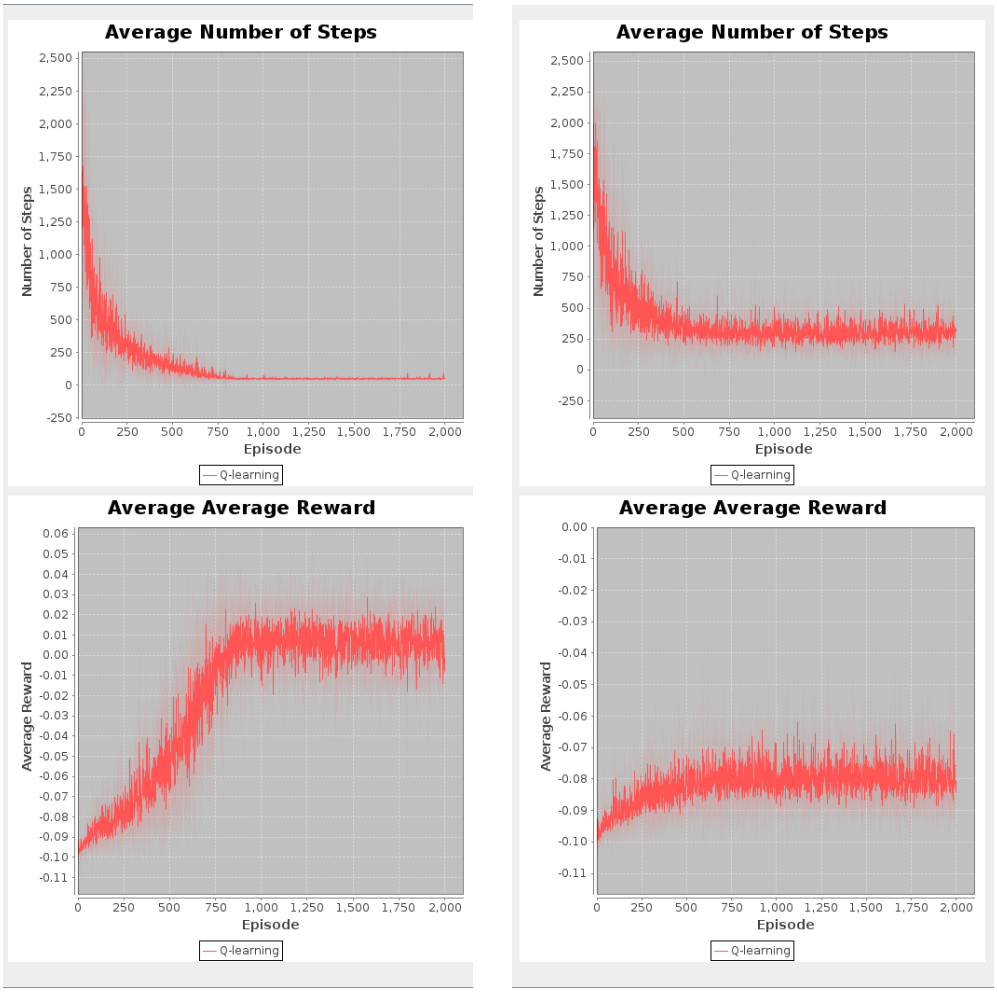

Figure 9: Graph of Q-learner performance on with epsilon of 0.05(left) and 0.8(right)

Conclusions

Markov decision process provide interesting problems. Value iteration and policy iteration produce quick and strong results given a lot of information. Value iteration is very helpful when the the actions are very deterministic as it is very fast and produces an optima solution. Policy iteration however is slower in general than value iteration but handles highly non-deterministic problems much better. This difference is due to the perspective the two algorithms take. Value iteration looks at the value of being in a given state while policy iteration of taking a certain action thus policy iteration handles when actions are non-deterministic much better. Q-learning on the other hand provides a method to produce strong results while being given very little comparatively. However, its exploration strategy heavily effected both in runtime and in quality of answer meaning the learner needs well tuned hyper parameters.

References

[1] Diao, E. (2016, May 17). MDPs and Reinforcement Learning [Computer software]. Retrieved April 22, 2018, from https://github.com/dem123456789/Machine-Learning/tree/master/MDPs and Reinforcement Learning

[2] MacGlashan, J. 2015. Brown-umbc reinforcement learning and planning (burlap). http://burlap.cs.brown.edu/. Accessed: 2015-07-30.