Nikolay Goncharenko, CS-7641-O01

Assignment #4

1. MDPs

1.1 Introduction

Two versions of a Grid World maze-like MDPs were created for the project:

- Small World: 5x5, 20 unique states

- Large World: 20x20, 354 unique states

The worlds look relatively similar in order to trace the effect of changing number of states. See page #3 for pictures of the worlds.

Each world has start state where agent is placed at the start of each learning episode. Once agent reaches the goal state, an episode ends.

Maze-like MDPs has a multitude of practical applications, starting from creating superior bots for computer games to controlling robots, drones, or self-driving cars. Maze also can represent a sequence of states that aren't necessarily depict transitions in a physical world. The agents learn to act optimally in 2D space in order to maximize discounted cumulative reward by reaching a goal state. The walls play role of obstacles that learning agents must avoid to act optimally. Such conditions satisfy requirements for controlling bots and robots in real-life environments and are used in a state-of-the-art research.

1.2 General Settings of MDPs

Settings used: Discount Rate = 0.9. The agent may take transitions in 4 directions: up, down, left, right unless there's a wall. Transition Probability = 0.9 (0.033 any of the other three directions). Reward for each transition = -1. Goal state is defined as a "state from which the agent deterministically transitions back to the same state with a reward of zero [1]". If the agent attempts to enter the wall state, it stays at the previous state with a reward of -1.

Since each action's reward = -1, we can use number of steps as cumulative negative reward. The lower the number of steps to the goal, the higher the cumulative reward and the lower cumulative negative reward. Therefore, the lower number of steps means the higher cumulative reward.

2. Testing

2.1 Comparison of Value Iteration (VI) and Policy Iteration (PI)

2.1.1 Convergence Criteria

For VI, convergence condition is $\Delta V(s) < 0.001$ that is the change in Value Function is less than 0.001 between iterations. For PI, convergence condition is reaching the same policy as VI algorithm that is not changed anymore in the following iterations. If we use the same criteria convergence as for VI, PI takes 2 iterations more that doesn't significantly increase a running time neither for Small, nor for Large Worlds.

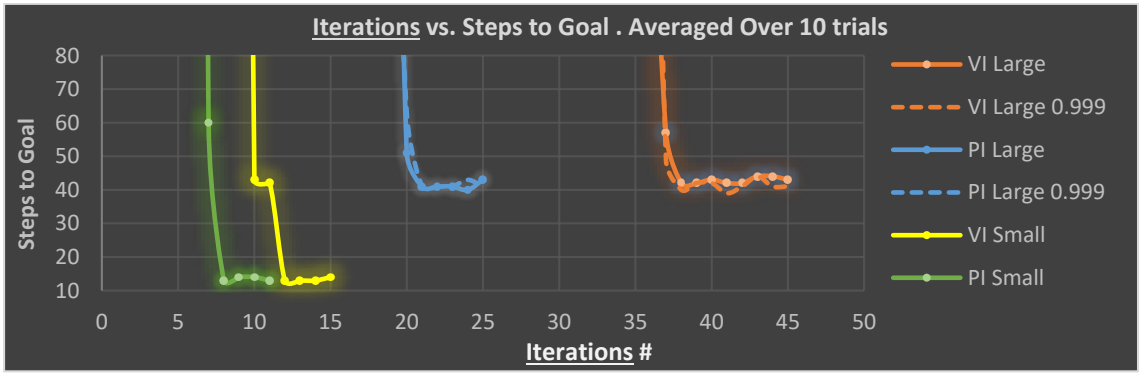
In addition, to the described above criterion, a graphical representation of convergence was also used. In graphical representation, we can consider convergence criteria as reaching maximum cumulative reward by an agent that isn't changed significantly between iterations, or in other words reaching minimum number of steps to a goal that isn't changed much between iterations. Since agent doesn't use all the states in the world to reach a goal state, even non-optimal policy that is locally optimal (agent path to the goal) may cause reaching minimum number of states before reaching a global optimum policy. Therefore, an overall policy must be estimated in addition to draw a conclusion about convergence.

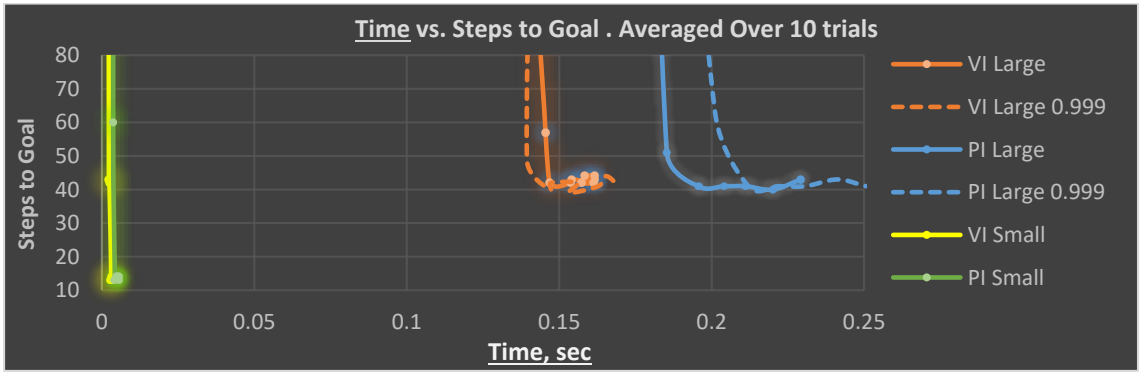
2.1.2 Analysis of VI and PI performance

Table 1. Comparison of VI and PI convergence

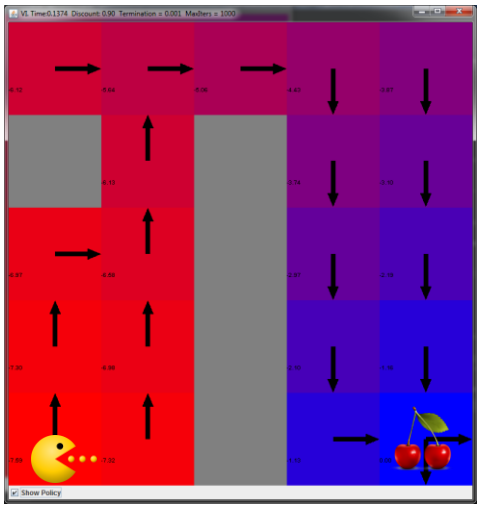
World	Discount Rate	Algorithm	Iterations*	Time, s	Time/iter,s	Time/Iter Growth, times
Small	0.900	VI	15	0.0032	0.00021	-
		PI	10/20*	0.0055	0.00055	-
	0.999	VI	16	0.0034	0.00021	-
		PI	10/20*	0.0056	0.00056	-
Large	0.900	VI	40	0.1542	0.0039	18.4
		PI	22/44*	0.2041	0.0093	16.9
	0.999	VI	43	0.1654	0.0038	18.3
		PI	23/46*	0.2308	0.0100	17.9

*For Policy Iteration (PI), number of iterations: #Policy Iterations/#Policy Evaluations

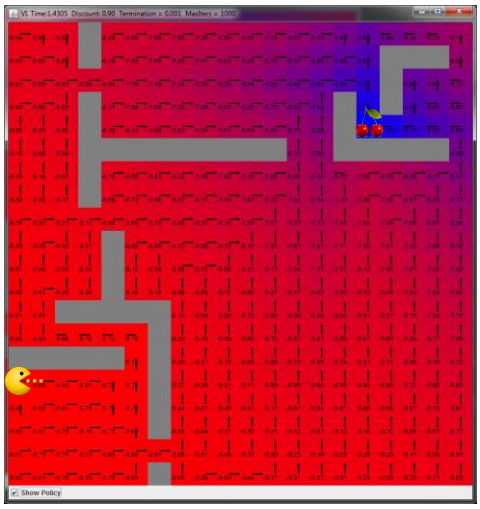




Pic.1. π PI and VI for Small World*



Pic.2 π PI and VI for Large World*



Discount Rate. Increasing discount rate from 0.900 to 0.999 (11%) doesn't significantly change running time for both algorithms (Table 1). However, according to Littman et al [2], for VI algorithm, "the number of iterations required can grow exponentially in the discount factor (Condon, 1992); as the discount factor approaches 1, the decisions must be based on the results that happen farther and farther into the future." Significant growth in running time is not seen in the current experiment. One reason might be relatively too low number of states to see the described effect of growth of the discount rate.

Running Time. Although PI takes 1.5 to 2 time less iterations than VI, PI converges approximately 1.3 times slower for Large and 1.7 time slower for Small Worlds compare to VI algorithm (Table 1 and plots above). PI internally uses modified VI update algorithm to approximate a Value Function under current fixed policy and considers only one action $\pi_i(s)$, corresponding to a fixed policy π_i that is tested (Policy Evaluation). VI algorithm also approximates Value function for the next iteration using Value functions of the successor state taken from the previous iteration. However, in contrast to PI, VI then implicitly computes policy by taking max over all actions. Because VI implicitly computes policy maximizing over all actions, it may save some time, omitting explicit computation of $\pi_i(s)$ at each iteration that is a 2nd part of PI algorithm called Policy Extraction.

VI: Value Iteration Update Rule

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$

PI Part 1: Policy Evaluation

$$V_{k+1}^{\pi_i}(s) = \sum_{s'} T(s,\pi_i(s),s')\left[R(s,\pi_i(s),s') + \gamma V_k^{\pi_i}(s')\right]$$

PI Part 2: Policy Extraction

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')]$$

As a result of similarity, both VI and PI algorithms converge to the same optimal policy π* but PI takes more time to converge. This optimal policy for Small and Large Grid Worlds is shown above (Pic. 1 and Pic. 2)

2.1.3 Effect of Number of States

According to Littman et al [2], per-iteration cost of VI = $O(|A||S|^2)$ or faster. Growth in number of states S from Small World to Large World is 17 times. From the Table 1 (see above), it follows that the rate of growth is approximately linear for VI (18 times).

According to Littman et al [2], per-iteration cost of PI = $O(|A||S|^2 + |S|^3)$. $|S|^3$ term is valid for the case when PI implements Policy Evaluation by solving system of linear equations instead of approximating Value Function values using Value Iteration Update Rule (above). Since we use VI, we should exclude $|S|^3$ term, so per-iteration cost of PI = $O(|A||S|^2)$. From the table 1, we see that per iteration cost is linear in number of states S for PI. For both VI and PI per-iteration cost doesn't reach worst-case scenario of $|S|^2$ probably due to nature of the problem and relatively small sizes.

2.1.4 Probability (Error Rate)

Setting transition probability to 0.6 changes the optimal policy for both PI and VI. The states have much lower utility values compare to transition probability set to 0.9 because the agent moves in the optimal direction only 60% of the time and reward becomes more distant. Time to converge increases when transition probability decreases. For deterministic model, optimal policy also changes. The states adjacent to the goal now have values of -1.00 because probability of transition probability to goal state is 100% when the optimal action is taken. Utility values increase for deterministic case because every action result in intended state, so goal state becomes closer.

2.2 Q-Learning

In real world situations, there's often no model and/or reward function provided, so an agent should learn an optimal policy directly from the experience. In Q-Learning, agent samples the environment by performing an action, observing successor state *s'*, and corresponding reward *r*. Each sampling is used to update current approximation of action-value function Q(s, a).

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

To control update of Q(s, a), learning rate α is introduced. Learning rate controls weight of the latest sampled value $r + \gamma \max_{a'} Q(s', a')$ on overall estimate of an optimal Q(s, a). In order to converge to an optimal Q(s, a), learning rate should be slowly decreased, so the weight of last samples in overall approximation decreases. However, setting α to a constant low values also may work well.

Another important hyper-parameter is epsilon (ε) that controls exploration/exploitation tradeoff. Agent choses action *a* greedily based on max Q(s', a') with probability of (1- ε) and chooses action randomly with probability ε. High values ε result in exploration of the environment by acting non-greedily but finding new, possibly better paths to the goal. ε = 0 results in a greedy behavior that brings maximum cumulative reward.

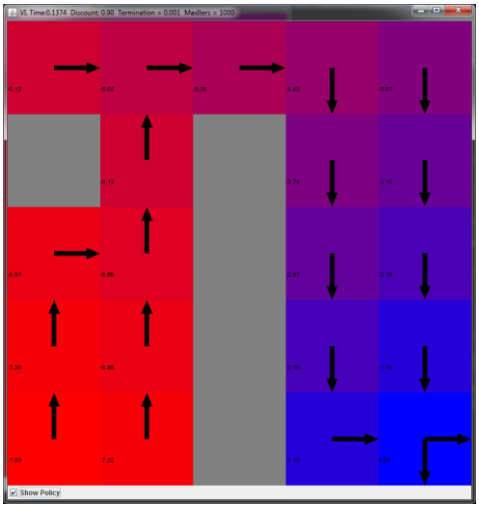
Since Q-Learning algorithm assumes that there is no known underlying model and reward function, it takes more iterations and time to reach an optimal policy compare to VI and PI algorithms. The best settings that lead to an optimal policy include slow decrease of learning rate and epsilon during a tested number of iterations for Small/Large Worlds. In the experiments, learning rate and epsilon were decreased in the range α = [0.9 to 0.001] and ε = [0.9, 0]. ε is decreased 10% with each 10% of iterations, and the last 10% of iterations performed with ε = 0. High values of ε at the beginning allow the agent to explore the environment because agent selects next action greedily with low probability and randomly with high probability. However, to converge to an optimal policy, ε should be decreased down to 0 at the end. Otherwise, the agent select transition with max Q(s',a') with probability at most of (1- ε_{lowest}) where ε_{lowest} is the lowest value of ε at the time the algorithm was stopped. Therefore, to get an optimal and well approximated path from the start state to the goal state, the agent should use exploitation first and moving greedily at the end. When the environment is well explored, greedy choices become optimal choices because Q(s',a') is well-approximated for all the possible transitions. The best results were achieved when ε = 0 for the last 10-20% iterations.

Initial Q(s, a) is set to 0.0 for all states. Since each action's reward is -1, all the Q(s, a) values, except goal, will be at most -1 by the last iteration. Therefore, setting initial Q(s,a)=0.0 is considered optimistic and encourages algorithm to explore in the beginning. No matter what action at first selected, it is going to have a lower reward compare to alternative adjacent actions still set optimistically; this forces the algorithm to explore all the adjacent unexplored actions as well [3].

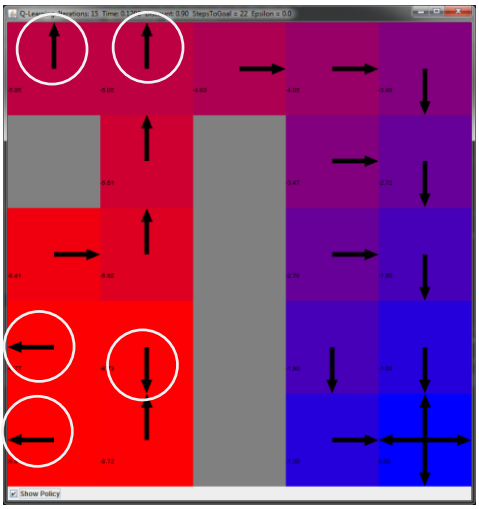
2.3 Comparison of Q-Learning VI, and PI

2.2.1 Small World

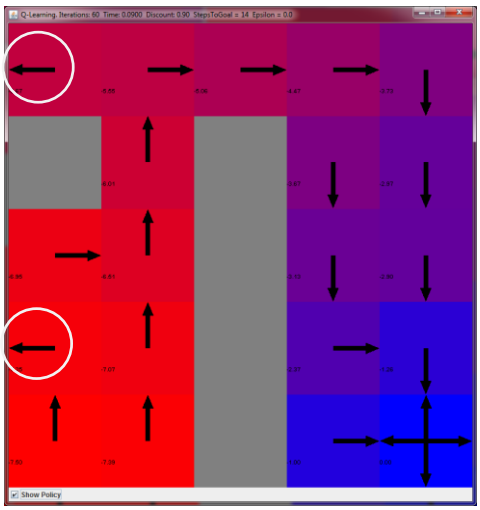
Pic.3 π obtained by VI (15 iter) and PI (10 iter)*



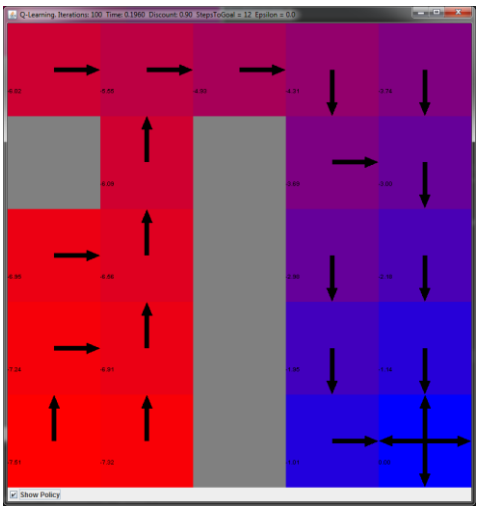
Pic. 4 Q-Learning π after 15 iterations (22 Steps)



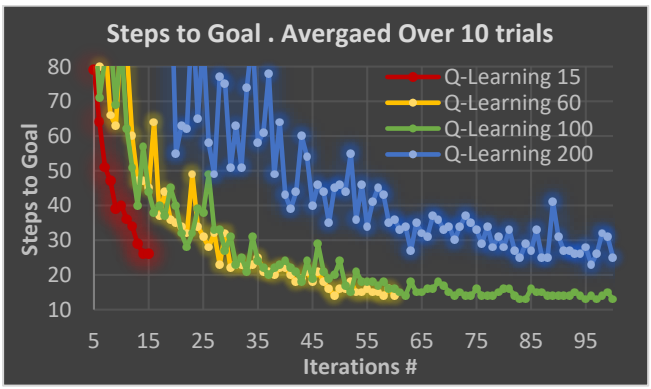
Pic 5. Q-Learning π after 60 iterations (14 Steps)



Pic 6. Q-Learning π after 100 iterations (12 Steps)*

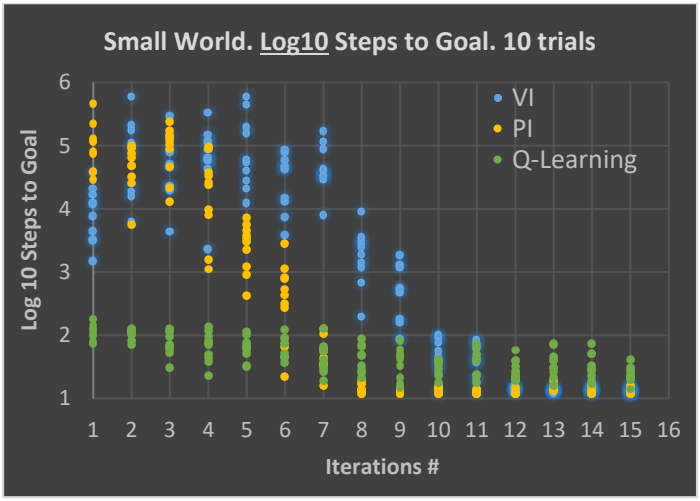


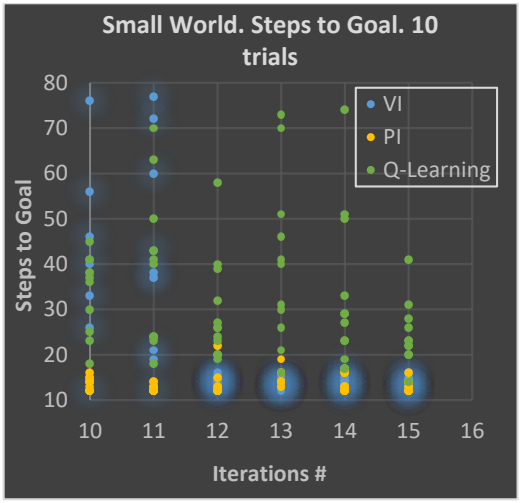
For Q-Learning, optimal policies start showing up at around iteration #60 although most of the generated policies are not optimal (Pic. 5) and the lowest number of steps to the goal is one step higher than for 100 and 200 iterations (right plot below). Generated policies become reliably optimal at around iteration #100 (Pic. 6), and sometimes the generated policies are the same as for VI and PI at # iterations = 100. Also, the lowest number of steps to the goal equals to the one for VI algorithm (13 steps). These both suggest that policy is converged globally (entire world) and locally (on the way from agent start state to the goal). Increasing number of iterations up to 200 doesn't produce much change in the policy and in lowest number of steps to the goal (see the right plot below). Therefore, algorithm is stopped at iteration # = 100 and considered converged to an optimal policy.

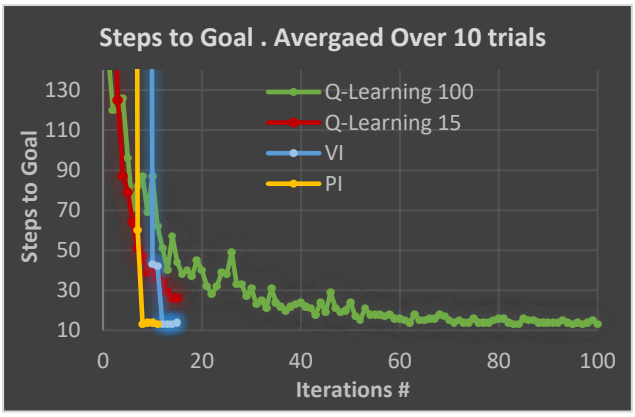


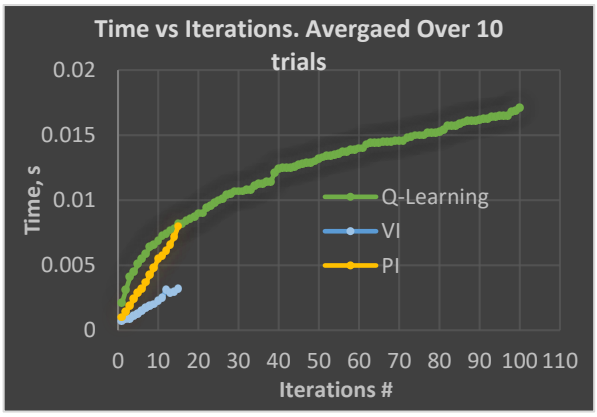
On the Log plot below (Steps to Goal), after the first few iterations, cumulative reward of Q-Learning is up to 4 orders of magnitude higher (Cumulative Negative Reward lower) than reward for VI and PI algorithms. The reason is that in VI, V(s) is updated recursively. Starting from a goal state reward is propagated to the start state with each iteration. After first iteration, all V(s) = -1 for non-stochastic model. This makes path to the goal completely random. In non-stochastic model, not yet propagated values of Q(s, a) become distorted by transition probability, making agent take a wrong path instead of just random path. In Q-Learning algorithm, at first iterations, agent takes actions 90% randomly starting from a start state and correct path is taken with probability slightly higher than random.

Starting at iteration #12, number of steps to the goal for PI and VI is in the range [12, 22] but for Q-Learning it is in the range [14, 74]. PI algorithm converges to an optimal policy at iteration #10 and VI converges to the same optimal policy at iteration #12. At iteration #15 Value Function of VI converges to the same values as PI Value Function. Q-Learning doesn't converge to an optimal policy even at iteration #15 (see Pic.3, 4 and plots above and below).



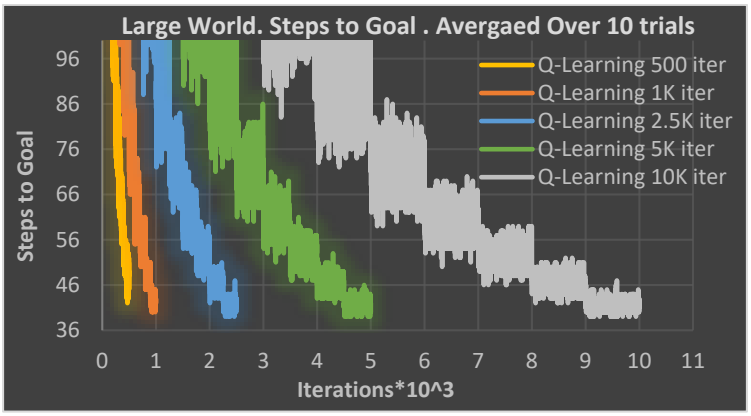




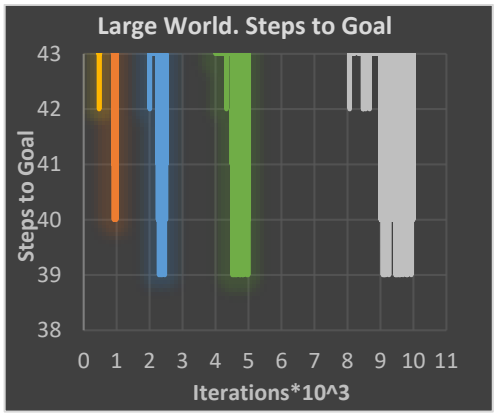


In terms of running time, Q-Learning is characterized by a fast growth in the first several iterations, but then growth in time significantly slows down. The possible reason is that at the start, there's no information about all the possible actions. Therefore, exploration of the adjacent states that can be reached from the given state takes additional time. VI and PI grow linearly with number of iterations from the very beginning because time spend on updates of V(s) and policy for each state are not changed with number of iterations.

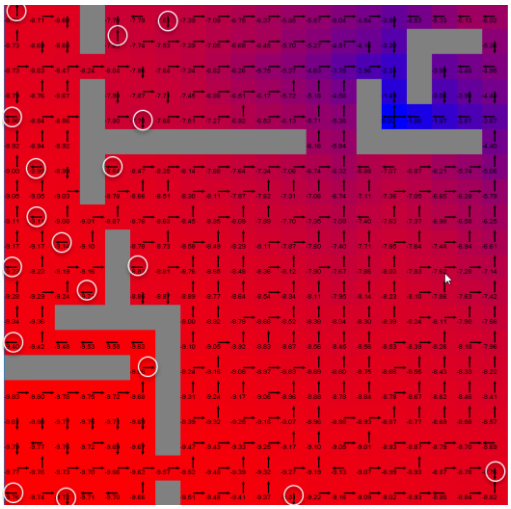
2.2.2 Large World



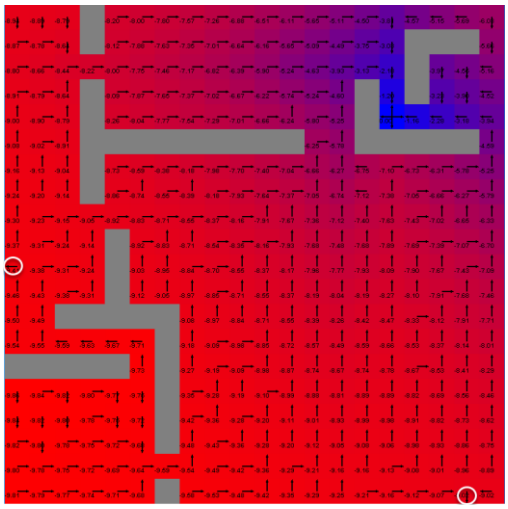
Pic. 7 Q-Learning π. 500 iterations (41 Steps)



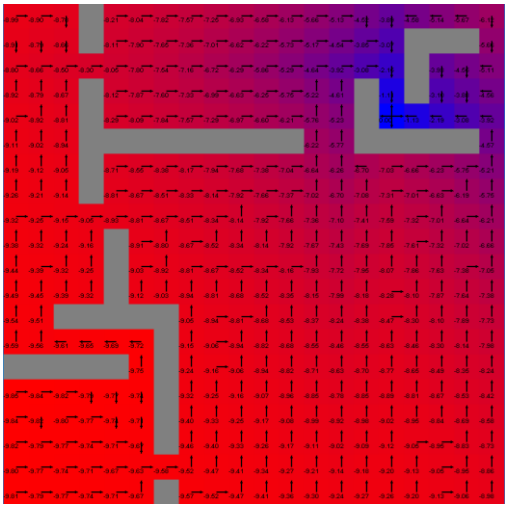
Pic. 8 Q-Learning π. 2500 iterations (39 Steps)

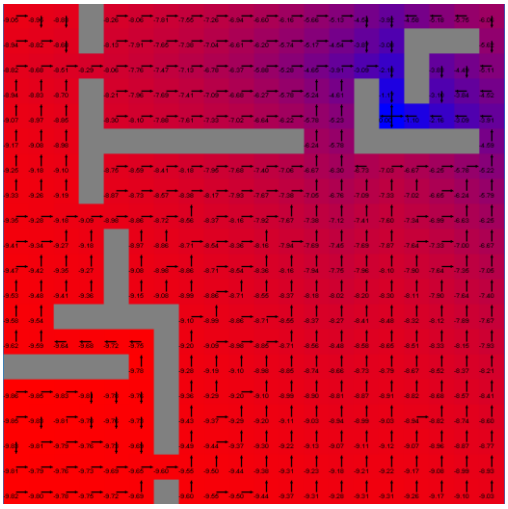


Pic.9. π Q-Learning for Small World. 5K iters*



Pic.10 π Q-Learning for Large World 10K iters*



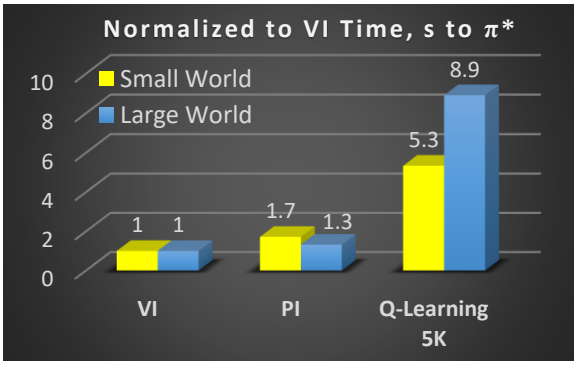


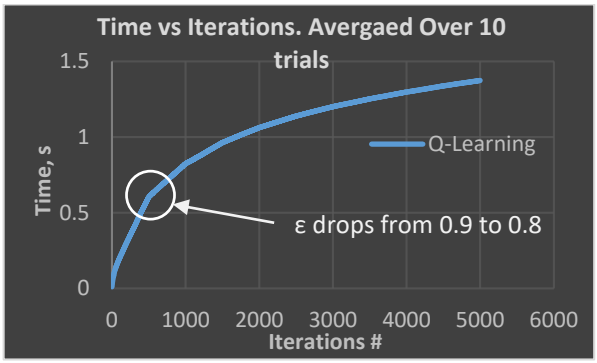
For Q-Learning, optimal policies start showing up at around 2500 iterations although most often the generated policies are not optimal (Pic. 8 above). However, the lowest number of steps to the goal

doesn't decrease anymore with increasing number of iterations (right plot above) and equals to the values reached by VI and PI algorithms. This means that the world isn't sufficiently explored during 2500 iterations although locally (from agent start state to the goal state) policy is converged. The fact that non-optimal states are always at the edge of the world confirms that the world needs to be better explored. Generated policies become reliably optimal at 5000 iterations (Pic. 9 above). Therefore, algorithm is stopped at iteration # = 5000 and considered converged. The generated optimal policies are not the same as for VI and PI even for the case of 500K iterations. The reason is that Q-Learning algorithm is promised to converge to true Q(s, a) values only when each state is visited infinitely often. The larger the World, the exponentially more iterations are required to visit each state sufficiently often to get all the Q(s, a) close enough to true values and get policy the same as produced by VI and PI.

Table 2 Comparison of Convergence to Optimal Policies

World	Algorithm	Iterations	Time, s	Time/iter,s	Time/Iter Growth, times	Iterations # Growth
Small	VI	15	0.0032	0.00021	-	-
	PI	10	0.0055	0.00055	-	-
	Q-Learning	100	0.0171	0.000171	-	-
Large	VI	40	0.1542	0.0039	18.4	2.67
	PI	22	0.2041	0.0093	16.9	2.2
	Q-Learning	5000	1.374	0.000275	1.6	50





From the histogram above follows that Q-Learning requires from 5 to 9 times more time to converge to an optimal policy compare to VI algorithm. The main growth in the time comes not from the growth of the time per iteration (that is 1.6 times) but from number of iterations. Q-Learning derives less information per iteration about V(s) than VI algorithm. For VI algorithm, Q(s, a) for all adjacent states are known, while Q-Learning algorithm can consider only one of the Q(s, a) per iteration by stepping to the adjacent state and approximating it, gaining information after the step was taken. Therefore, more iterations are required for Q-Learning to approximate action-value and state-value functions. From the right plot above, it's seen that as in case of Small World, most time-consuming part is the first 10-20% of iterations where algorithm continues randomly exploring all possible actions from a given state. This exploration strategy leads to suboptimal paths to the goal. Epsilon drops 10% each 500 iterations (right plot above), and, therefore, randomness in choosing an action also drops 10%, so path to the goal

becomes shorter. As expected, the main reason for increased running time of Q-Learning compare to the algorithms where model and reward functions are known is the need to approximate this unknown information by sampling the search space.

2.2.3 Additional Comparison to VI

To evaluate divergence of the Value function at each state, the sum of differences between Value function generated by Q-Learning and VI algorithm was calculated:

$$\Delta V(s) = \sum_s | V_{VI}^{\pi*}(s) - V_{QLearning}(s) | / |S|$$

This data (see plot below) confirmed that Q-Learning algorithm with selected number of iterations produced a good approximation of Value function produced by VI algorithm that used known true model and reward function.