CS7641 Machine Learning Assignment IV

Markov Decision Processes

Wang Lu, GTid: 903355610
29th October ,2018

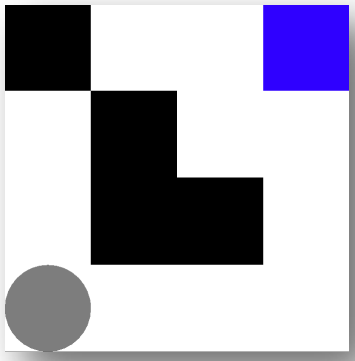
Choice of Problem

I chose the classical grid world problem for this assignment: in a simplified 2D grid world, given the destination and starting point, make the single agent learn the best way to reach the set goal. It has attracted great interest due to its real-world representing and abstracting power, as well as its application potential on future autonomous driving. At the same time, this problem has many derivatives depending on the specific set of rules of a concrete grid world example, which allows wider and deeper range of parameter tuning and exploration for dissecting purposes.

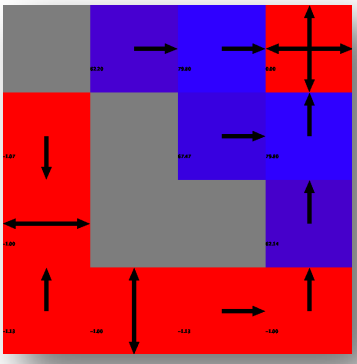
In particular, the first MDP problem, is about a square world with arbitrary but stationary map, where the agent moves stochastically from one corner towards another corner (where sits the goal) and tries to avoid crashes during the migration[1,2,3]. Notably, the agent is allowed to live forever in this world (infinite horizons). **One easy map and one difficult map are studied respectively.** The second MDP problem is extra exploration beyond the assignment description. However, **it makes meaningful comparisons with last part**, completes the exploration of epsilon and learning factor and deepens the understanding of Q learning. It is based on a more realistic setting: a 'taxi driver' attempts to safely and reliably transport passengers to their destinations in a grid network with similar 'city plan' to Manhattan streets, traffic lights on every cross road and a considerable flow of traffic[3]. However, the agent has only limited time for task. The world for second problem is deterministic, but goal position changes in every trial round.

Part I. Infinite Migration in a Stochastic Stationary Square World

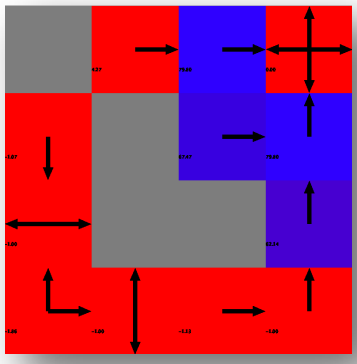
Easy Grid World Map (4x4)



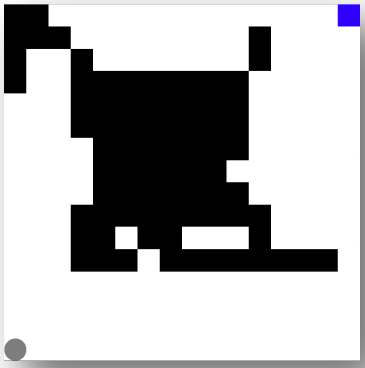
Value Iteration (iter1)



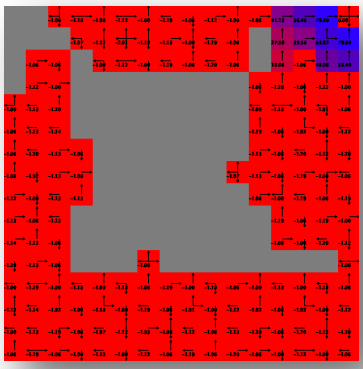
Policy Iteration (iter1)



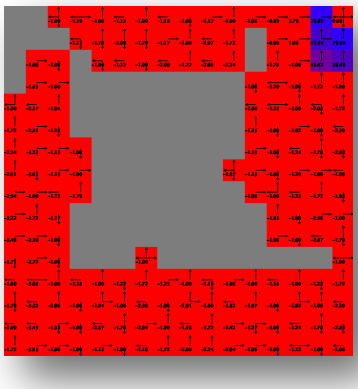
Hard Grid World Map (16x16)



Value Iteration (iter1)



Policy Iteration (iter1)



Map Characteristics and Problem Size

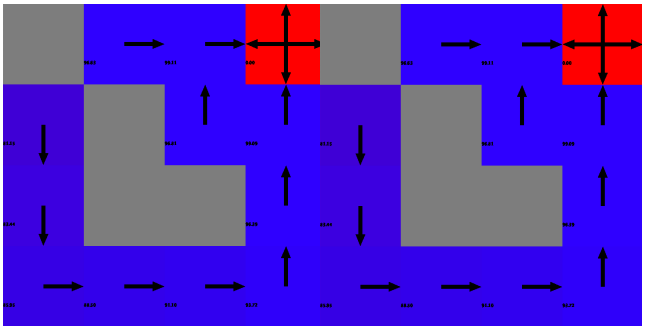
For this grid world, two different world maps are chosen. They resemble each other in the overall obstacle positions and route shapes, but the road margin of the second is modified and a few inaccessible cells and irregular cavities are created. The agent needs to learn policy for total 12 reachable states for map1 and 169 for map2, which is 14 times increased with regards of problem size.

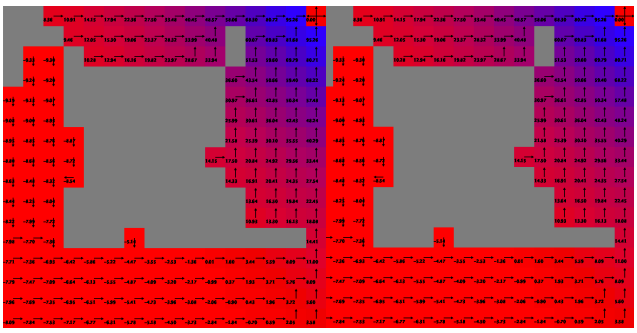
Physics of the world

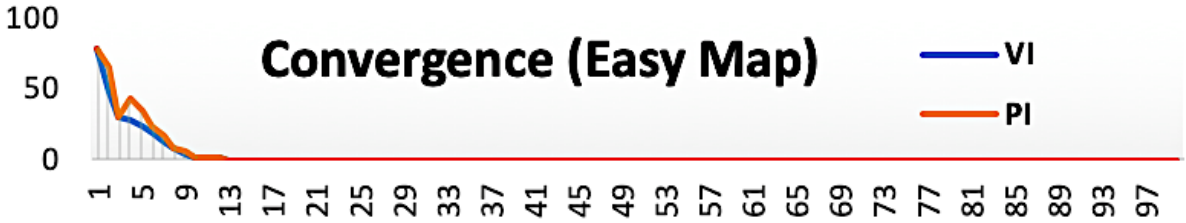
The goal always sits upper right corner (shown as blue cell on map) and the agent always start from bottom left corner (shown as grey dot on map). The agent is allowed only to take 'forward', 'downward', 'left' or 'right' action in each step. Diagonal traversing is illegal. However, the success of action is only probabilistic: 0.8 for the intended direction, and 0.0667 for the rest 3 directions. Entering the goal state gives reward of 100, while -1.0 for everywhere else unoccupied. Attempts leading to crashes against wall or obstacles would be punished. The agent doesn't have time limit in this world. The discount factor gamma to solve the 'Immortal Dilemma' is set constant as 0.99.

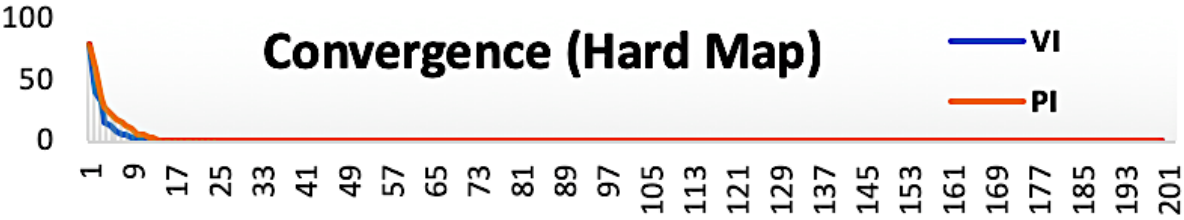
Value Iteration versus Policy Iteration

For both easy and hard cases, VI seem to backpropagate wider than PI at initial stage, as indicated by bigger short wavelength color area (high utility) at iteration 1. Unsurprisingly, the policies computed based on different utility are also quite different. However, they converge on same utility and generate the same policy after hundreds of iterations (heatmap below). In fact, they both converge es early as ~10 epochs. As indicated by the line chart below, VI converges smoother than PI, this is because VI adds old values from last iteration and new sample by ratio, while PI re-calculate the value by matrix manipulation every time.

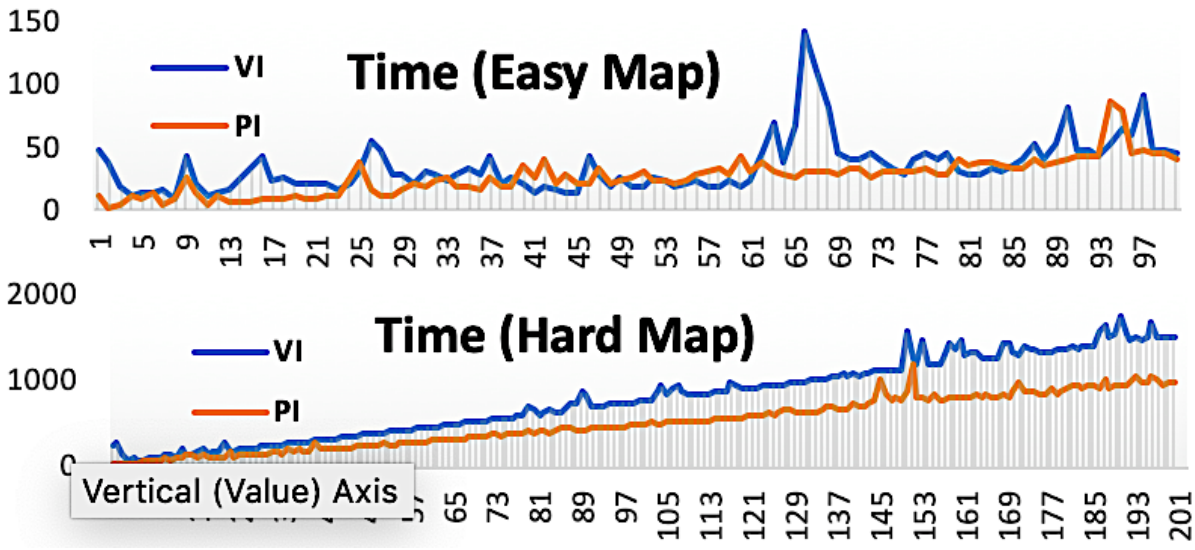








PI is in general more time efficient than VI, which might benefit from the fact that it depends on shrewd matrix manipulations rather than wrapped data structure. Consistent with intuition, their computational time is linearly correlated to iteration number. Average total reward of VI is 90.06, and corresponding number of steps is 11.94 (entering non-rewarding initial and goal state are also counted). The longer sequence of steps, the lower total reward for that round. Consistent with the fact that agent get punished by lingering over the world (-1. for every extra step).



How number of states affect things

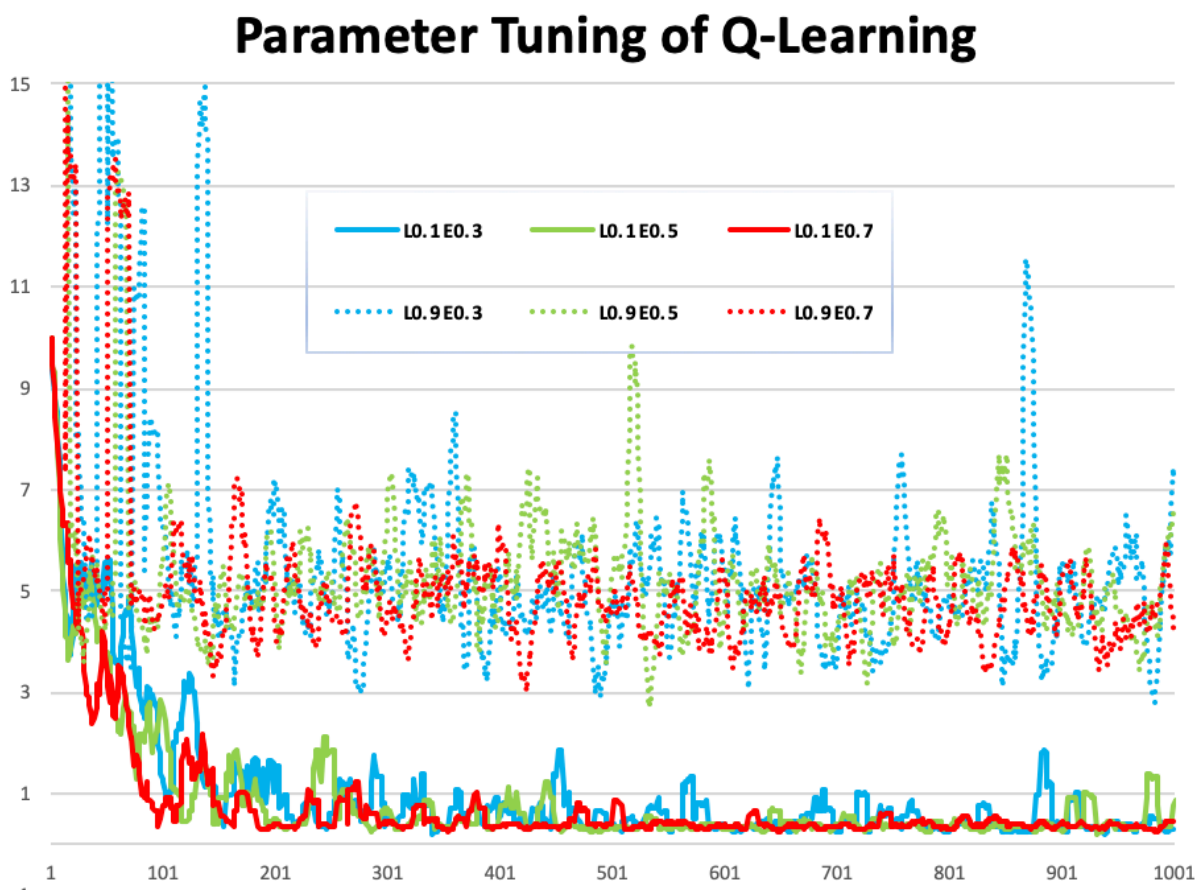
A very obvious effect: VI and PI run slower. With the same 100 iterations, VI take 17 folds of running time on the Hard Map than that of Easy Map, PI spent 12 folds, which are roughly the same with problem size increase. However at least in this example, the problem size didn't affect convergence, both VI and PI took around 10 iteration to converge on same answer. VI converges faster and smoother than PI.

Reinforced Learning on Easy Map

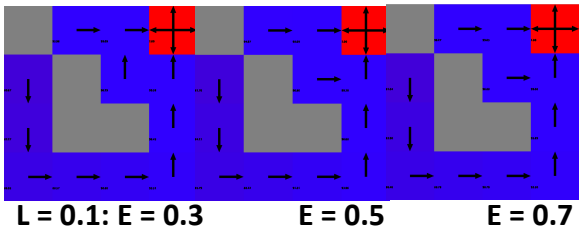
This time the agent knows neither about transition model nor the rewards. It has to figure out what is the best action for each single state by observing <<state, action>, Q>. Due to the agent's lack of domain knowledge, it requires many more rounds to learn and converge than VI and PI case. I chose 1000 as the max iteration and picked 2 learning rates (L, 0.1 and 0.9) and 3 exploration factors (epsilon/E, 0.3,0.5 and 0.7) and together make 6 combinations. Bigger learning rate means higher weight on new sample over past experience (more 'open minded'), experiments with learning rate equal to 0.1 are indicated by solid lines, 0.9 by dashed lines. Bigger epsilon means more explorative and adventures, more likely to take random decision rather than the best action by knowledge. Higher epsilon experiments are drawn with warmer color: blue: epsilon 0.3, green: epsilon 0.5 and red: epsilon 0.7.

Q-learning not just converges much slower, but also much less smooth than VI and PI. It occasionally makes bad decisions even at 1000 iterations. Groups with learning rate = 0.9 don't converge nor show clues of converging after 1000 iterations, indicated by the huge stand deviation of the curve. For both dashed lines and solid lines groups, red curve (epsilon

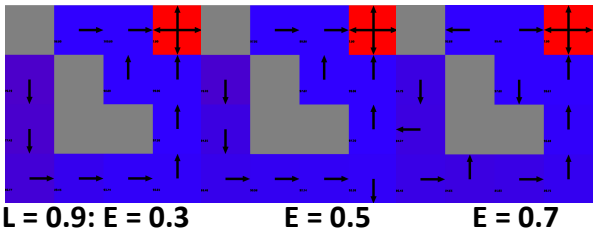
= 0.7) performs the best. In fact, L = 0.1, E = 0.7 is the only experiment setting gives convincing converging pattern after 1000 iterations here. Taking the results together, this problem favors a solver that cautiously picking new information but willing to take risk of exploration.



Indeed, the final utility and policy table generated by L = 0.1, E = 0.7 has the most similar look to that of VI and PI, and L = 0.1 group is much better than L = 0.9 group:



L = 0.1: E = 0.3 **E = 0.5** **E = 0.7**



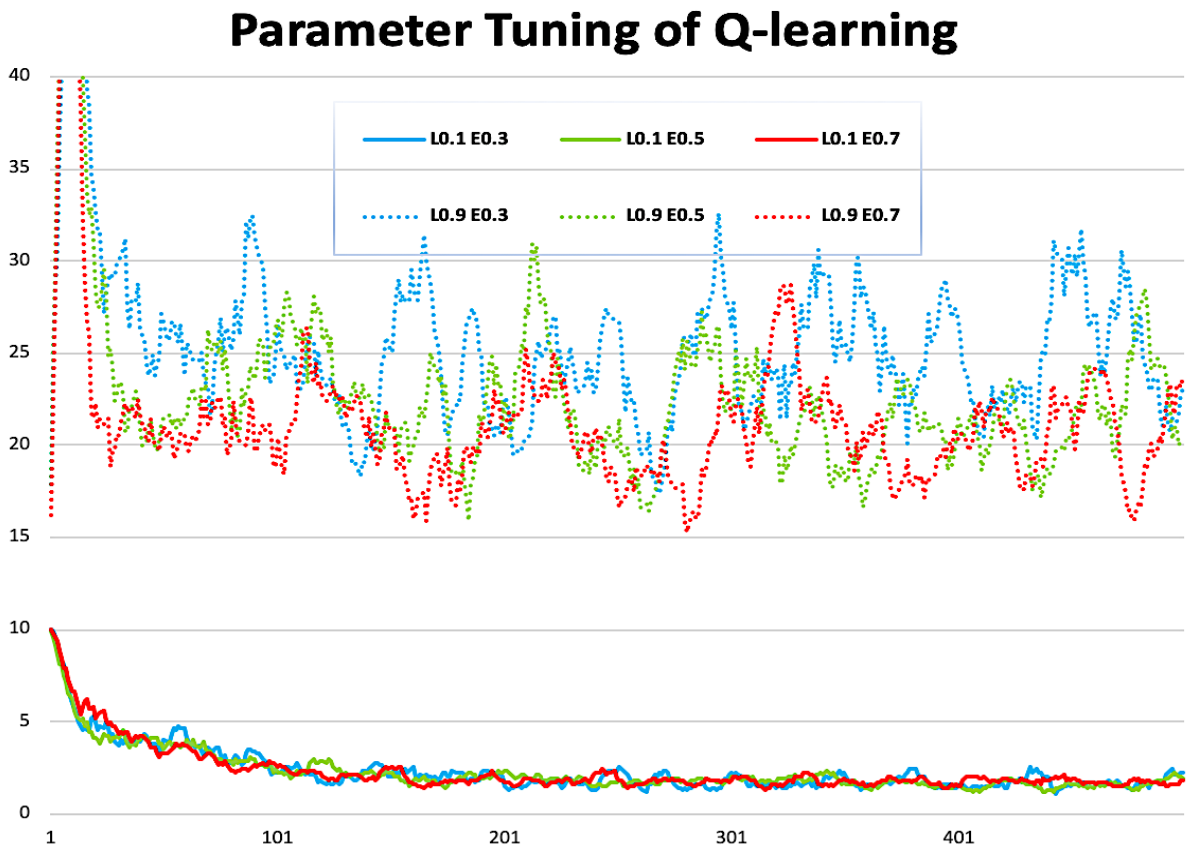
L = 0.9: E = 0.3 **E = 0.5** **E = 0.7**

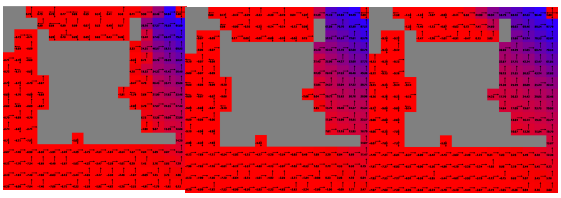
average:	time	reward	steps	outlier: steps
L0.1 E0.3	0.22033525	91.589	10.411	888
L0.1 E0.5	0.22927054	89.219	12.781	3249
L0.1 E0.7	0.2931105	42.346	59.553	50000
L0.9 E0.3	0.18698606	75.821	26.179	1036
L0.9 E0.5	0.24892121	70.909	31.091	1401
L0.9 E0.7	0.30142698	62.836	39.164	2031

Noticeably, the average scores of experiments are highly skewed by outliers. For example, average score of the best setting L = 0.1 and E = 0.7 is the worst, due to a early stage outlier, which hit the goal until 50000 steps.

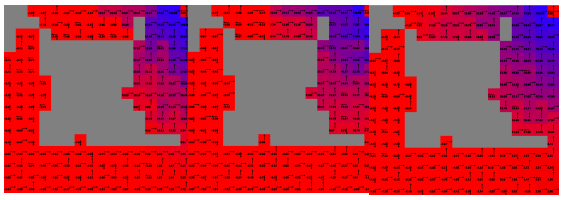
Reinforced Learning on Hard Map

Similar pattern is shown for hard map. However, this time the divergence between L = 0.1 and L = 0.9 group is even greater and differences within the group is smaller. This observation is confirmed by the heatmap generated after 500 iterations below.





L = 0.1: E = 0.3 E = 0.5 E = 0.7



L = 0.9: E = 0.3 E = 0.5 E = 0.7

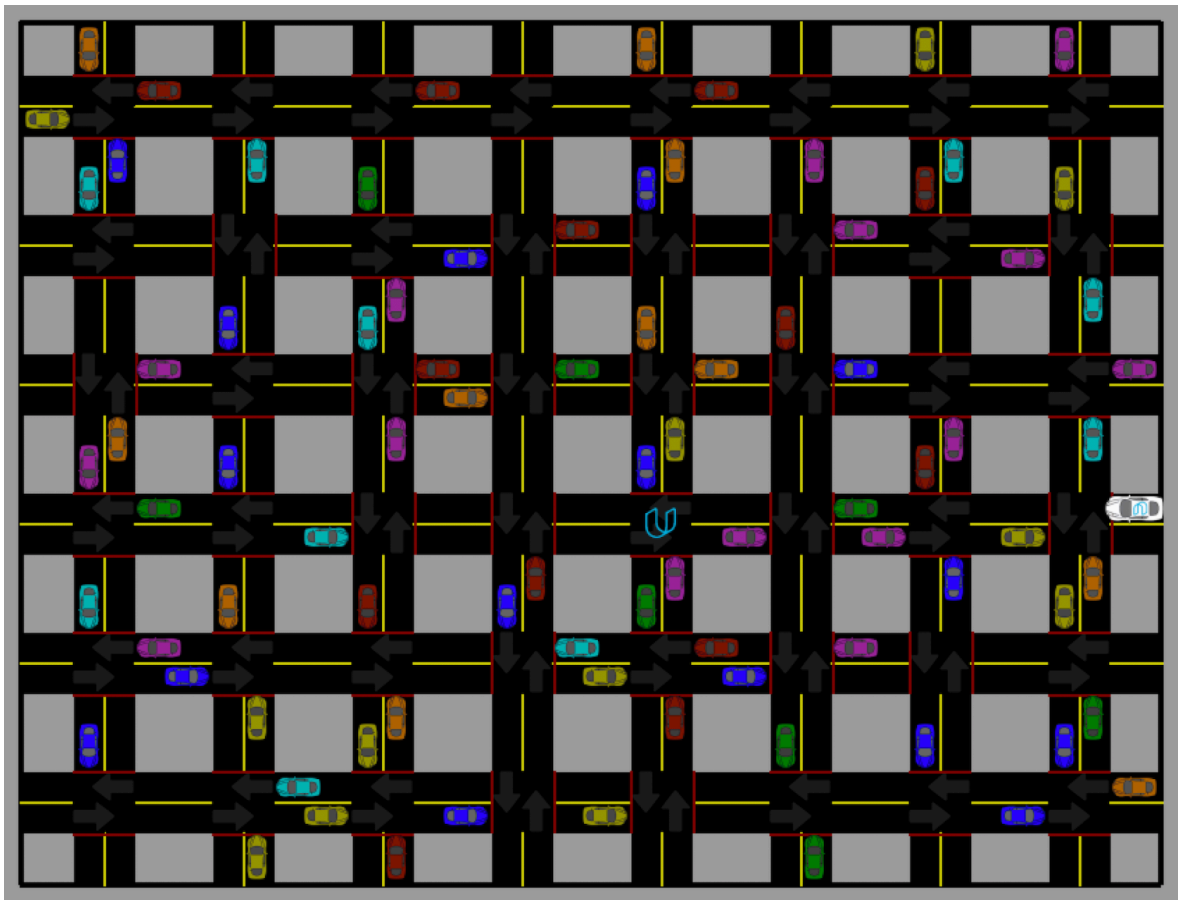
Part II. Self-Exploration:
Smart Cab in Dynamic Deterministic Grid Network

As mentioned earlier, problem of interest in part II is under a more realistic setting: In a simplified 2D world defined by a fixed 8x6 grid network, the driving agent (the only white car below) has to learn to drive imaginary passengers to their destinations without disobeying traffic rules. (The agent is not informed of the traffic rules, it has to learn from observing followed reward after taking some reaction under given states). This time, the agent is not confined to start from bottom left or end at top right, instead the initial and goal states are chosen randomly. In the example map given below, the agent start at cell (8,4) and is expected to reach goal cell (5,4). Another two key differences are the determinacy and finiteness of the world.

Map Characteristics and Problem Size

This is an 8x6 grid network. Each cell has one North-South direction road, one East-West direction road, one building (colored grey) and one traffic signal on crossroad (red lines

indicating the corresponding waypoint is blocked at that moment). 100 dummy drivers which drive randomly are scattered on the map (black, cyan, blue, green, magenta, orange, red, and yellow cars). The smart driving agent (learner) is the only white car on map. There are a lot of variables can be considered for decision making, making the state space not as straightforward as the last one. Due to the choice of features, the problem size can vary.



Physics of the world

Intuitively, the agent has to reach the goal in given time limit (here it's 5 times the Manhattan distance of the initial cell and goal cell). The agent can choose to move forward, turn left, turn right or idle, but move backward is illegal. The speed is fixed as 1 unit at each time unit, that's to say if the agent facing east moves forward from cell (8,4), next time unit it will reach (7,4). The agent should obey traffic rules and avoid accidents, just like what taxi driver do in real world. The key differences of this problem and a real-world self-driving problem are: 1) the movement is deterministic; 2) traffic flow is homogenous and map is symmetric; 3) Don't consider accelerating: speed is either the fixed value (1 unit per time interval) or zero (idle). However, it evolves from last static problem to a dynamic programming problem, that is the agent has to respond to changing environment, which makes it more interesting and emphasizes the 'learning' aspect.

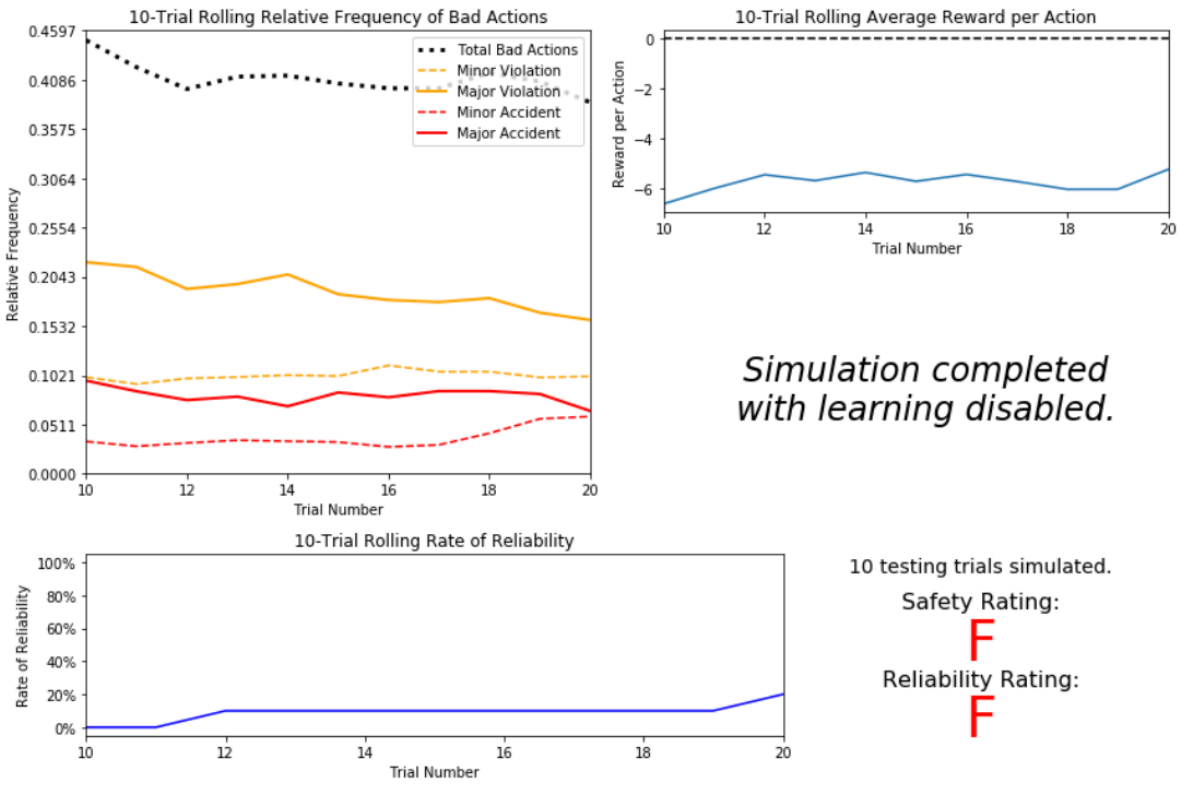
Based on the two main expectations towards the agent, two evaluation metrics are proposed to quantitatively assess agent performance, that is *Safety* matric and *Reliability* matric. A letter grade A+ is given for Safety matric only when agent commit no traffic

violations and always choose the correct action in all trials; A letter grade A+ is given for Reliability only when agents 100% reaches destination before deadline. A, B, C, D, F is given correspondingly to worsening performance. Detailed standard is as below:

Grade	Safety	Reliability
A+	Agent commits no traffic violations, and always chooses the correct action.	Agent reaches the destination in time for 100% of trips.
A	Agent commits few minor traffic violations, such as failing to move on a green light.	Agent reaches the destination on time for at least 90% of trips.
B	Agent commits frequent minor traffic violations, such as failing to move on a green light.	Agent reaches the destination on time for at least 80% of trips.
C	Agent commits at least one major traffic violation, such as driving through a red light.	Agent reaches the destination on time for at least 70% of trips.
D	Agent causes at least one minor accident, such as turning left on green with oncoming traffic.	Agent reaches the destination on time for at least 60% of trips.
F	Agent causes at least one major accident, such as driving through a red light with cross-traffic.	Agent fails to reach the destination on time for at least 60% of trips.

Make Random Action: When the agent not Learning

The simplest situation where the agent naively takes random actions from [‘Forward’, ’Left’, ’Right’, ’None’] is first studied. No learning is implemented at this stage.



You can about 4% and 9% of the total actions respectively cause minor or major accidents, together making 0.13 accident rate. The driving agent received about -6.0 on average for each of the action it took. Considering the positive rewards agent received by making correct decisions, which took up 60% of the total actions, were averaged out by the other 40% of bad decisions so much which gives this low mean reward, I believe the driving agent has been penalized heavily. Given the fact that the agent just drives randomly, it's not surprising for it to get very low reliability score as well. At this stage the smart cab is definitely neither safe nor reliable for passengers: given a medium-short distance driving which takes, says 5 actions, the calculated accident rate would be more than 50%; for medium-long -distance driving taking more than 10 actions, only less than 25% probability that this passenger would reach safely. It would be a disaster if such an agent is allowed to drive. Similarly, only one fifth of trials are completed, most of the time the agent just loops around and gets lost. It's definitely not a reliable driver either.

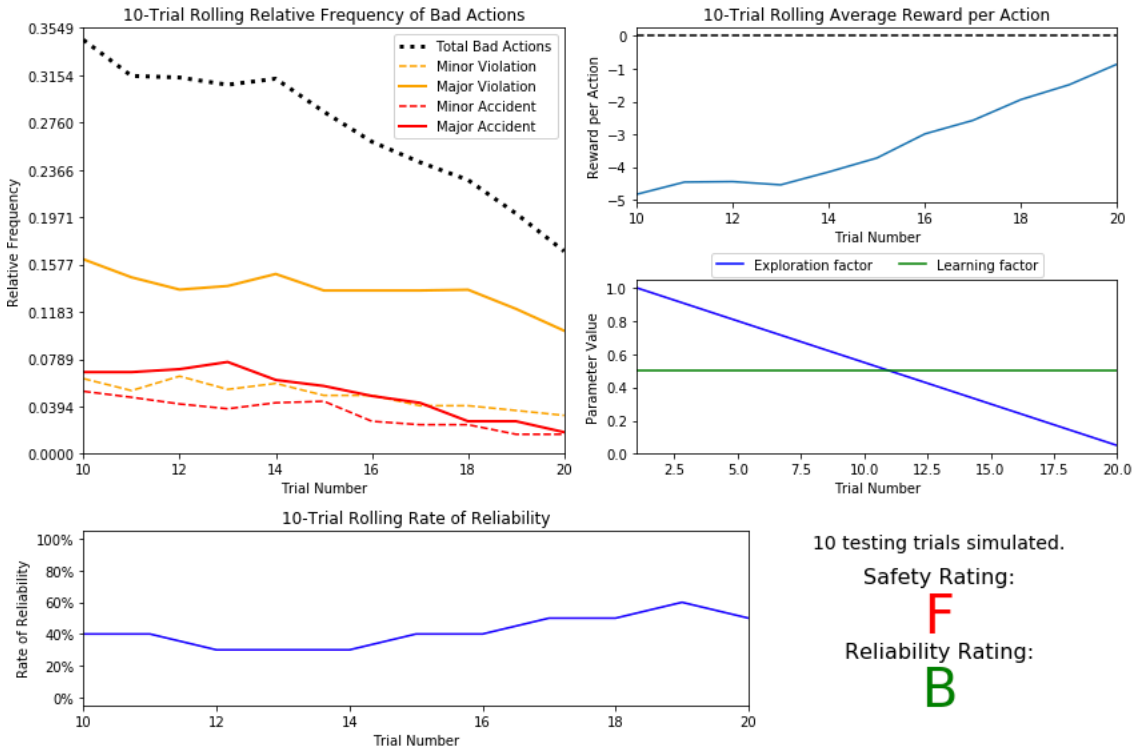
Inform the Agent: Define state space

This is a relatively complicated world. A lot of features can be candidates for composing the state. For example, the initial position, the goal position, the deadline, the instant orientation, the traffic light, oncoming cars and so on. However, it's unwise to include all of them as the state space would grow astronomically and make it impossible task to cover each state enough times at reasonable training time. Only the minimum set of input should be considered. And the chosen features are as follows:

Features	corresponding variable	relevancy	possible values	number of values
direction to destination (waypoint)	self.next_waypoint	efficiency	[None, 'forward', 'left', 'right',]	4
traffic light	inputs['light']	safety	[True, False] (green and red light respectively)	2
intention of the oncoming car	inputs['oncoming']	safety	[None, 'forward', 'left', 'right',]	4
intention of the immediate left car	inputs['left']	safety	[None, 'forward', 'left', 'right',]	4

In total 4x2x4x4 = 96 states. Considering the mathematical expectation of minimum driving distance to be 6.0 units, on average there would be at most only 6*5=30 times per trial to cover states. Thus 20 trials would be able to roughly iterate each state 20*30/96 ~ 6 times. And for 100 trials that would be 30. Quite feasible numbers for reinforced learners. I didn't choose `deadline` as it has too many possible values to be modeled on. Introducing this feature will expand the state space by 12 times ((8-1) + (6-1) = 12). Same reason for excluding goal position or initial position, they expand the space state by 8x6 times. Noticeably, although intention of oncoming and immediate left car is considered as important state feature, the intention of the immediate right car is ignored since the traffic in the grid world follow right-hand rule. Right turn as a small turn is unlikely to cause accidents especially when all dummy drivers in this simulated world obey traffic rules.

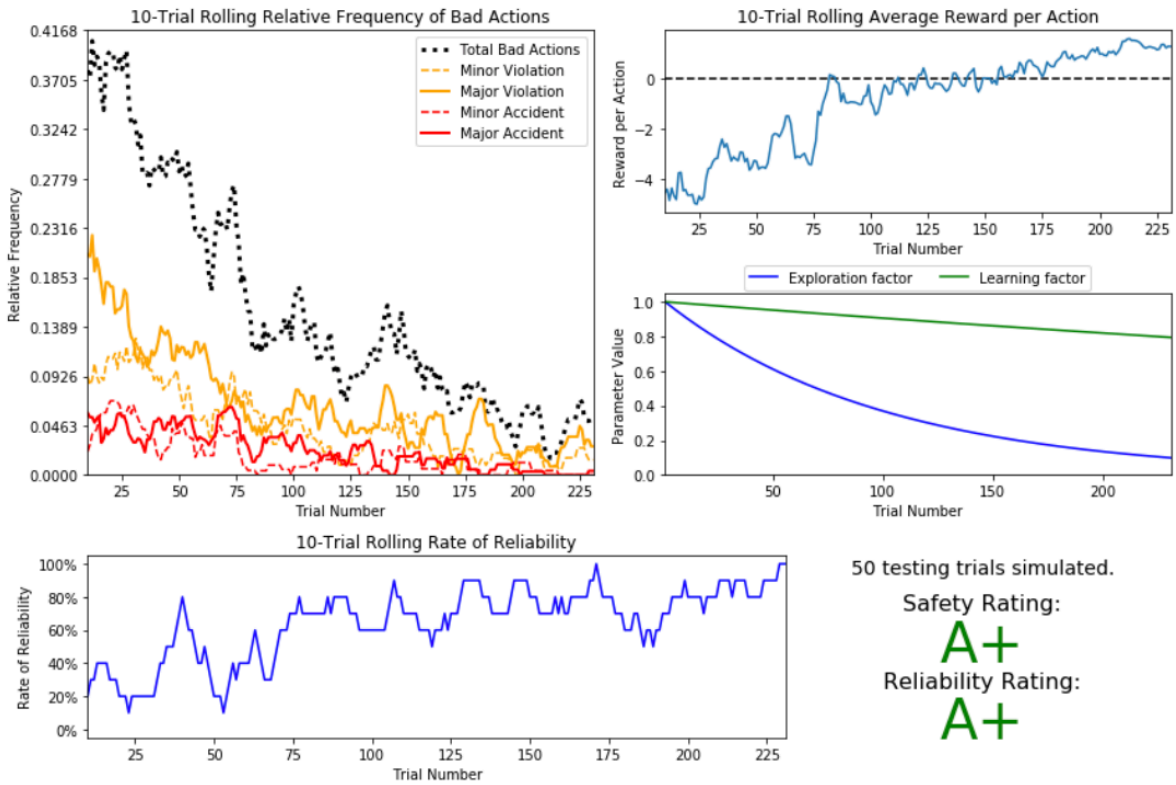
After informing the agent about the current state [‘direction to destination (waypoint)’, ’traffic light’, ‘oncoming car’, ’immediate left car’], and asking it to make decision only at epsilon chance random, and at 1- epsilon chance choose the best action to its knowledge, the Safety and Reliability score have significantly improved. As the learner starts to use its past experiences in making informed actions. However, note that epsilon decays linearly at exact 0.05 each trial, after 20 trials it stops exploration and testing trails are applied for evaluation purpose.



The default Q-learning agent still sometimes goes in loop. But bad actions drop a lot as the number of training trials increased, especially for actions lead to accidents. The average reward increase from nearly -5.0 for 10 trials to -1.0 for 20 trials. The most significant improvement is on Reliability rating, which has improved from F to B. But the safety rating is still quite bad since the agent didn't manage to avoid major accidents. To improve this, more training trials can be included. In theory, the agent can be allowed to learn for an incredibly long periods of time; however, another goal of Q-Learning is to transition from experimenting with unlearned behavior to acting on learned behavior. For example, initial high epsilon decaying very slow (extreme situation epsilon = 1 and never decays) will certainly make it learn, but never let it act, so that the agent would behave as the naïve driver in last section. A very small epsilon or epsilon decays too fast on the other hand, makes the agent rigid and misses the chances of learning. Same story for learning factor. Here the old knowledge and new knowledge is weight by constant ratios. Choosing shrewd decaying functions for the exploration factor and learning factor are another important consideration in improving the Q learning agent.

Improved Q-Learning Agent

After changing the decaying function of epsilon to 0.99**trials_number, learning factor to 0.99 – trial_number, and allow the agent to train more than 200 rounds, the total bad actions especially major accidents leading actions are pushed backward and backward to 0, which benefits the agent by lifted average reward per action. In the end the agent achieves A+ for both safety and reliability based on 50 testing trials.



Comparison between Part I and Part II Reinforced Learning Problem

Unlike part I where epsilon and learning factor are fixed throughout the learning process, this problem gives a flavor of how exploration and learning factors can affect the convergence behavior and policy quality of Q learning in grid world problem. Noticeably, discount factor gamma is not used. This is due to few differences of this problem compared to last part problem. First, introducing of gamma leads to a map with decreasing contour Q values propagate from the destination. However, the destination in this problem keeps changing, which would mess up the learning output across trials. Secondly, the limitation from state space. Since the world is more complicated and dynamic, state space is composed of not positions like the last problem, instead the agent was informed only about the immediate status to take flexible response. Including gamma means feeding the agent with relative coordination information, which immediately enlarges the state space by 5*7=35 times. This is not just computational unaffordable but also unnecessary, as `waypoint` already contains most of useful information and almost 9 times compressed than coordinates.

References:
[1] Burlap
[2] Juan J. San Emeterio's Grid World Problem code
[3] Jonathan Tay's Grid World Problem code
[4] Previous Project from Udacity's Machine Learning Engineer nanodegree