CMPSCI 687 – Fall 2017 — Final Project

Evaluating Average-Reward Reinforcement Learning on the Product Delivery Domain

Pankaj Bhambhani, SPIRE ID 30566626

Abstract

Reinforcement Learning has a lot of practical applications outside of building computer game bots (cha [2017]) ranging from Manufacturing, Inventory Management, Product Delivery Management (routing of multiple delivery vehicles) and Finance (optimizing trading strategies) among others. In this Project, we look at one such application, the domain of Product Delivery and see how Reinforcement Learning fares here. We used Average-Reward Reinforcement Learning (ARL) here instead of Discounted Reinforcement Learning as we found that better suited to this task. In particular, we used Average-Reward Q-Learning and a variant called After-State Q-Learning (ASQ). In the coming sections, we illustrate the problem of product delivery domain and represent it as a Markov Decision Process (MDP). We then present the Average-Reward Q-Learning algorithm and illustrate its applicability here. We also present a Non-Learning Greedy Agent as a benchmark to compare the performance of our learning agent. Finally, we present the experimental results (plots) for a couple of variants of the problem.

The Product Delivery Domain and its MDP Representation

For this Project, we used a version of product delivery domain based on Proper and Tadepalli [2006]. As mentioned in that work, this domain is a combination of different aspects of vehicle routing problem and inventory control problems. The problem involves delivery of a single product to multiple destinations using several agents (trucks in this case). The source of the product is a depot location that is connected (may be indirectly) to a series of shops, which are the final destinations. Often reaching a shop from the depot involves passing through a sequence of locations. Each shop has a specific inventory level, which may decrease over time (simulating customer purchases). The goal of the agent is to maintain a non-zero inventory level at each of the shops (i.e. to ensure no shop runs out of product) and do so using the minimal number of total moves.

To represent this problem as a Markov Decision Process (MDP) $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, d_0, \gamma)$, we design the states, actions, transitions and rewards as follows:

- $\mathcal{S}$ is the set of States.
  - Here, we represented a state $s = (\mathbf{p}, \mathbf{i}, \mathbf{l})$ as a tuple of three vectors, $\mathbf{p}$ is a vector containing the position of each truck, $\mathbf{i}$ is a vector containing the current inventory level of each shop, and $\mathbf{l}$ is a vector containing the current load of each truck.
  - The location of each truck can be an integer value between 1 to $|L|$ where $|L|$ is the number of locations in the network (including shops, depot and any intermediate nodes that the truck must pass through).

  - Each shop can have an inventory that is discretized into level $1, 2, \ldots, m_I$ where m_I is the max Inventory that any shop can hold (we assume all shops can hold inventories upto m_I).
  - Similarly each truck load is discretized into level $1, 2, \ldots, m_L$ where m_L is the max Load that any truck can hold (again we assume all trucks can hold loads upto m_L).
  - If the number of agents (trucks) is $|A|$ and the number of shops is $|S|$, then the total number of possible states in this arrangement could be $(|L|^{|A|})(m_I^{|S|})(m_L^{|A|})$. As can be seen, the number of states is exponential in the number of trucks and shops, illustrating the curse of dimensionality. For a not-too-large problem of 4 trucks, 5 shops and 10 locations with the max Inventory and max Loads being 5 each, the number of possible states (i.e. size of state space) is $(10^4)(5^5)(5^4) = 19531250000$.
  - For this project, we use a tabular representation of state space which is obviously not suited to higher state-space dimension. Hence, we only consider smaller variants of the problems (with the max number of trucks being restricted to 2, more on this later). Proper and Tadepalli [2006] presents a scalable representation of a state space using Tabular Linear Functions (TLF). TLFs are similar to normal linear function approximations with the function represented as a linear of combination of features $\phi_i(s)$, each multiplied by a weight θ_i. In addition, each weight θ_i is a function of other features (referred to as "nominal features"). The representation of a value function for such a state could be as shown below:

$$V(s) = \sum_{i=1}^{n} \theta_i(f_{i,1}(s), f_{i,2}(s), \ldots, f_{i,m_i}(s))\phi_i(s) \tag{1}$$

    where $f_{i,1}(s), f_{i,2}(s), \ldots, f_{i,m_i}(s)$ are the nominal features for the weight θ_i

- $\mathcal{A}$ is the set of actions.
  - Each truck can do three types of actions -
    1. it can move in one of the four directions (up, down, left, right).
    2. It can unload x number of units of product. ($x \in 1, \ldots, m_L$)
    3. It can wait (do nothing) (we explain below the purpose of this action).
  - The total number of possible actions is $|a| = 5 + m_L$. For our project, we kept $m_L = 4$ for all experiments, hence the total number of actions is 9.
  - The size of action space (the number of joint actions) is again exponential in the number of the agents, illustrating another curse of dimensionality. If the number of trucks is $|A|$, the size of action space is $|a|^{|A|}$. In our case, if we have $|A| = 4$, the size of action space if $9^4 = 6561$.

- $\mathcal{P}$ is the transition matrix of next states, given the current state and the joint action.
  - In our case, the effects of the agent's actions on the environment are deterministic, however the environment itself has stochastic processes running in the background (the consumption of products by customers at each stop is modeled as a stochastic process running independently of other shops).
  - Thus, the possible next states given a current state and action depends on the number of shops. In fact, it's exponential in the number of shops, illustrating the third curse of dimensionality.

4 locations				**10 locations**			
T = 5		**T = 20**		**T = 5**		**T = 20**	
4 locations 3 shops T = 5 1 truck	4 locations 3 shops T = 5 2 trucks	4 locations 3 shops T = 20 1 truck	4 locations 3 shops T = 20 2 trucks	10 locations 5 shops T = 5 1 truck	10 locations 5 shops T = 5 2 trucks	10 locations 5 shops T = 20 1 truck	10 locations 5 shops T = 20 2 trucks
Plot - Figure 3	Plot - Figure 4	Plot - Figure 5	Plot - Figure 6	Plot - Figure 7	Plot - Figure 8	Plot - Figure 9	Plot - Figure 10

Table 1: Summary of Different Experiment Variants

  - In our experiment, we model the customer consumption in a simple manner as follows - for each shop s_i, we reduce the inventory level by 1 after $T * i$ units of time have passed, where T is some constant indicating the periodic nature of the process. We use two different values of T, i.e. $T = 5$ and $T = 20$, resulting in two variants of the problem.
  - Finally, the next state also depends on the truck location, shop inventory level and truck load. If the truck is at a location other than a shop (including the depot), then any of the unload actions will have no effect. Similarly, a truck can only unload when it's current load is non-zero. And the amount unloaded cannot be more than the truck load or the max additional amount the shop can store (since the inventory cannot go above the max inventory level m_I). The trucks are also loaded automatically upon reaching the depot.

- $\mathcal{R}$ is the reward function, which depends on the current state, the action and the next state.
  - In our experiment, every agent move action (i.e up, down, left, right) incurs a penalty cost of -0.1 (mimicking fuel cost).
  - In addition, at every time-step, a penalty of -5 is given for each shop that has zero inventory.
  - The wait action is introduced so that if an agent does not need to fill up any shop inventory, it can avoid moving around randomly and avoid the -0.1 fuel cost.
- $d_0(s)$ is the distribution of initial states. In our experiments, the initial state is deterministic - all the agents start at the depot location, with max load m_L, and all shops are fully loaded (inventory $= m_I$).
- γ is the reward discount factor. We used $\gamma = 1$ in all our experiments.

We experiment with the two values of T, along with 2 values for the number of agents (1 and 2), and two different location maps - one with 4 locations (3 shops and 1 depot) and other taken from Proper and Tadepalli [2006] with 10 locations (5 shops and 1 depot). Figure 1 and 2 show the two location maps. This resulted in 8 different experiments over all. Table 1 summarizes the different variants.

Methods - Average Reward Q-Learning, After-State Q-Learning (ASQ) and the Non-Learning Greedy Agent (NLGA)

In this section, we introduce the learning agent we used to solve the product delivery problem. We give a brief overview of Average-Reward Learning, then introduce our learning agents - Q-Learning

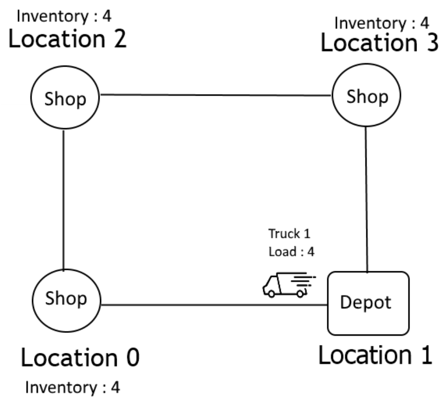

Figure 1: Location Map with 4 locations, 3 shops. (Inventory and truck loads are sample values)

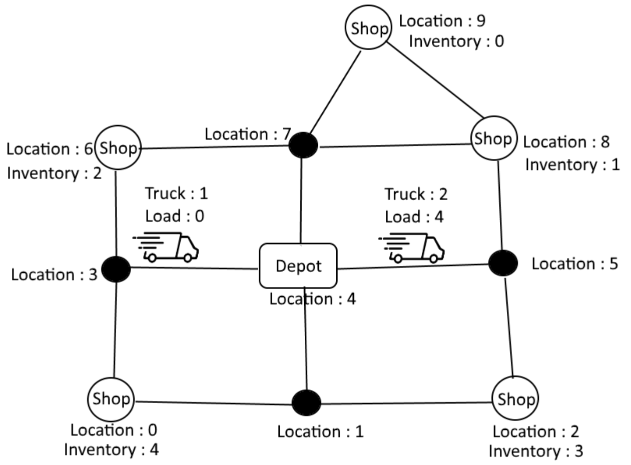

Figure 2: Location Map with 10 locations, 4 shops. (Inventory and truck loads are sample values)

and After-State Q-Learning. Finally, we briefly mention a non-learning greedy agent which we built as a benchmark to compare the performance of our learning agent against.

Average-Reward Reinforcement Learning

Our understanding of Average-Reward Reinforcement Learning (ARL) is based on Proper and Tadepalli [2006], Proper and Tadepalli [2009], Mahadevan [1996], Tadepalli and Ok [1998] and Tadepalli et al. [1994]. An Average-Reward MDP is similar to the usual MDP (called the discounted reward MDP) except that the goal is to maximize the average expected reward per time-step, which is to referred to as *gain* ρ. For a given policy π and a start state s_0, we calculate this by taking the expected value of the sum of rewards obtained till the N^{th} time-step and taking the average in the limit.

$$\rho^{\pi}(s_0) = \lim_{N\to\infty} \frac{1}{N}\mathbb{E}[\sum_{t=0}^{N} R_t | s_0, \pi] \tag{2}$$

The goal of ARL is to obtain a policy π^* which gets as close to the optimal gain ρ^*. However, the maximal average reward gained could depend on the start state (since the start state may not always allow transition to a state that fetches the optimal average reward in the long run). Hence, for average reward learning problems [Mahadevan, 1996] suggests to define a "biased" or "relative" value function which is basically the sum of rewards earned relative to the gain ρ.

$$V^{\pi}(s) = \lim_{N\to\infty} \mathbb{E}[\sum_{t=0}^{N} (R_t - \rho^*) | s_0, \pi] \tag{3}$$

Using the Bellman equation, we can calculate the optimal Value Function in this case as follows:

$$V^*(s) = \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s')(R(s,a,s') + V^*(s')) - \rho^* \tag{4}$$

and the corresponding Q-function is as follows:

$$\mathcal{Q}^*(s,a) = \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a,s') + \max_{a' \in \mathcal{A}} \mathcal{Q}^*(s',a')) - \rho^* \tag{5}$$

Average-Reward Q-Learning

For a given state, action, next-state and reward sequence (s_t, a_t, s_{t+1}, r_t), we calculate the TD-error δ_t in this case as follows:

$$\delta_t = r_t + \max_{a' \in \mathcal{A}} \mathcal{Q}(s_{t+1}, a') - \rho - \mathcal{Q}(s_t, a_t) \tag{6}$$

And the corresponding TD-update is

$$\mathcal{Q}(s_t, a_t) = \mathcal{Q}(s_t, a_t) + \alpha * \delta_t \tag{7}$$

where α is the learning-rate hyper-parameter for the $\mathcal{Q}$ function.

In our experiments we use ϵ-greedy policy for action selection. The gain ρ itself needs to be estimated and is done using TD-learning. However, the gain is updated only when the action taken is a greedy one. The TD-update for gain is calculated as follows:

$$\rho = \rho + \alpha_\rho * \delta_t \tag{8}$$

where α_ρ is the learning-rate hyper-parameter for gain ρ. Algorithm 1 shows the pseudocode for Average-Reward Q-Learning

Algorithm 1 Presudocode for Average-Reward Q-Learning

```
1: procedure Q-LEARNING($s_0, t_{max}$ )
2:     $s \leftarrow s_0$
3:     $t \leftarrow 0$
4:     while $t < t_{max}$ do
5:         Find an action $a_{max} \in \mathcal{A}$ that maximizes $\mathcal{Q}(s,a)$
6:         Take an exploratory action or the greedy action $a_{max}$. Let $a$ be the action taken, $r$ be
   the reward and let $s'$ be the resulting state.
7:         if greedy action was taken then
8:             $\rho = \rho + \alpha_\rho(r + \max_{a' \in \mathcal{A}} \mathcal{Q}(s',a') - \rho - \mathcal{Q}(s,a))$
9:         $\mathcal{Q}(s,a) = \mathcal{Q}(s,a) + \alpha(r + \max_{a' \in \mathcal{A}} \mathcal{Q}(s',a') - \rho - \mathcal{Q}(s,a))$
10:        $s \leftarrow s'$
11:        $t \leftarrow t + 1$
```

After-State Q-Learning

For our project, we also tried a variant of Q-Learning called After-State Q-Learning which helps deal with the stochastic state-changes that are independent of the agent's action. This is loosely based on the After-State H-Learning described in Proper and Tadepalli [2006]. In the general Q-Learning case, calculating the TD-error would require stepping through all the possible next states to calculate the maximal Q-value. Since, the number of next states is usually exponential in the number of parameters (number of shops in this case), this computation can be quite expensive.

To avoid this, the agent only steps through the next states which are possible due to the agent's actions, ignoring those which are possible only due to the stochastic nature of the environment. We did this in our pseudo-code by letting the environment also supply the after-state, i.e. a state without the random inventory deductions due to customer actions. We used this as a next state to our Q-Learning agent.

Hill Climbing for Action Space Search

We also used the simple form of Hill Climbing method described in Proper and Tadepalli [2006] to speed up the computation of the joint action. Here, every joint action $\mathbf{a}$ is represented as a vector of sub-action, each by a single agent, i.e. $\mathbf{a} = (a_1, \ldots, a_k)$ where k is the number of agents. Each agent action in the vector is initialized to "wait" action. Then, starting at a_1, we consider all possible action values of the current agent i (other than the current value) and set a_i to be the best action. We repeat this for $a_2, \ldots, a_k$. The process then starts over at a_1 and repeats until $\mathbf{a}$ has converged to a local optimum (no agent actions changed during a single pass over actions). We also do something similar to the approach of Homework 4, where if there are multiple possible best joint actions, we select uniformly among them.

Non-Learning Greedy Agent

We also present a simple hand-coded non-learning agent as a benchmark to compare the performance of the two learning agents. This agent follows a greedy approach, and is loosely based on the greedy agent described in Proper and Tadepalli [2006]. Given a state, it tries to identify candidate shops that need the product (in our case it's just shops whose inventory value goes below the max Inventory m_I). Given this candidate set, for each agent it tries to identify a shop that is closest to that agent and farthest away from all the other agents (i.e. one where only this agent can reach with minimum penalty). Other tweaks include waiting if all shops have max inventory, unloading if currently on a shop location, or heading to depot if agent load is zero.

Experimental Results and Conclusions

We conducted the various experiments summarized in Table 1. For average-reward learning, the experiment runs as a single long episode without ever resetting the environment. We followed the test method used by Proper and Tadepalli [2006], running the experiment for 10^6 time steps, divided into 20 phases of training and evaluation. For each phase, we ran 48,000 steps of training followed by 2,000 steps of evaluation. During the evaluation phase, the hill climbing search was turned off and complete search over the action space was used instead. For each experiment, we ran 30 trials and recorded the mean return for each time-step. We report the results for the three agents described in the above section. For each experiment we show the corresponding plot of Online-Average Reward (shown as moving average) vs the number of iterations (time-steps). Wherever possible, we've tried to minimize the window length without introducing too much variance in the plot. Table 1 list the plot figures corresponding to the experiments. We also show an approximate running time taken by the three agents (see Table 2). Note that even though we show the runtime of the NLGA agent for 30 trials for comparison, in most cases it was only run for 1 trial since it's actions are always deterministic.

We tried a few combination of hyper-parameters for both the learning agents, in the end we found the following values to work best

- the learning rate $\alpha = 0.1$

Agent	Execution Time (s) for 30 trials
Q-Learning	255
ASQ	248
NLGA	58

Table 2: Execution Times for Different Agents for 30 trials

- $\epsilon = 0.1$
- learning rate $\alpha_\rho = 1$ and decayed as $\alpha_\rho = \frac{\alpha_\rho}{\alpha_\rho + 10}$

In general, we found that a slower learning rate (like $\alpha = 0.001$) gives lower average reward and a higher learning rate (like $\alpha = 0.5$ or 1) leads to divergence. We also found that higher values of ϵ (particularly 0.5) worked really well for a smaller number of iterations (time-steps) but did not give good results as the number of time-steps increased.

The overall results showed that the two learning agents almost match the performance of the non-learning greedy agent when the number of location is small and the number of trucks is just 1. However as the number of trucks increases or the number of location increases or the value of T decreases (indicating shops get empty more quickly), the learning agents don't perform as well as the greedy agent. We believe this can be accredited to the size of the state-space and the fact that we used a tabular representation.

As for the two learning agents, they produced similar results in most cases, with the Q-Learning being slightly better in some cases and the ASQ being better in other cases. We also found that ASQ has lower variance in the average reward (graph is smoother) as compared to Q-Learing agent.

Future work could include using linear approximations to represent the Q-functions as outlined by Proper and Tadepalli [2006]. A neural network based approach is also worth trying. We believe these two approaches could improve the average reward gained by the two learning agents, especially as the number of agents increases (see Egorov [2016] for solving of multi-agent deep RL problems and Busoniu et al. [2010] for general solving of multi-agent RL problems).

References

Reinforcement learning and its practical applications, Apr 2017. URL https://chatbotsmagazine.com/reinforcement-learning-and-its-practical-applications-8499e60cf751.

Lucian Busoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. *Innovations in multi-agent systems and applications-1*, 310:183–221, 2010.

Maxim Egorov. Multi-agent deep reinforcement learning. 2016.

Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine learning*, 22(1):159–195, 1996.

Scott Proper and Prasad Tadepalli. Scaling model-based average-reward reinforcement learning for product delivery. In *ECML*, volume 6, pages 735–742. Springer, 2006.

Scott Proper and Prasad Tadepalli. Solving multiagent assignment markov decision processes. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 681–688. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

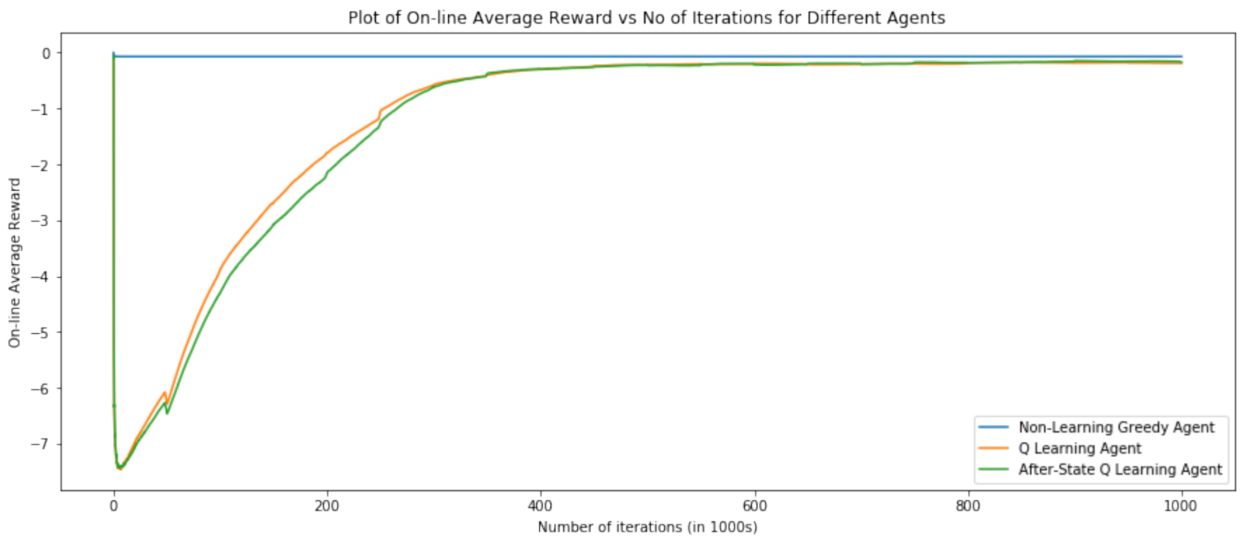

Figure 3: Plot of On-line Average Reward vs No.of Time-steps (30 Trials, 10^6 time-steps, 4 locations, 3 shops, 1 truck, T = 5)

Prasad Tadepalli and DoKyeong Ok. Model-based average reward reinforcement learning. *Artificial Intelligence*, 100(1):177–224, 1998.

Prasad Tadepalli, DoKyeong Ok, et al. H-learning: A reinforcement learning method to optimize undiscounted average reward. Technical report, Corvallis, OR: Oregon State University, Dept. of Computer Science, 1994.

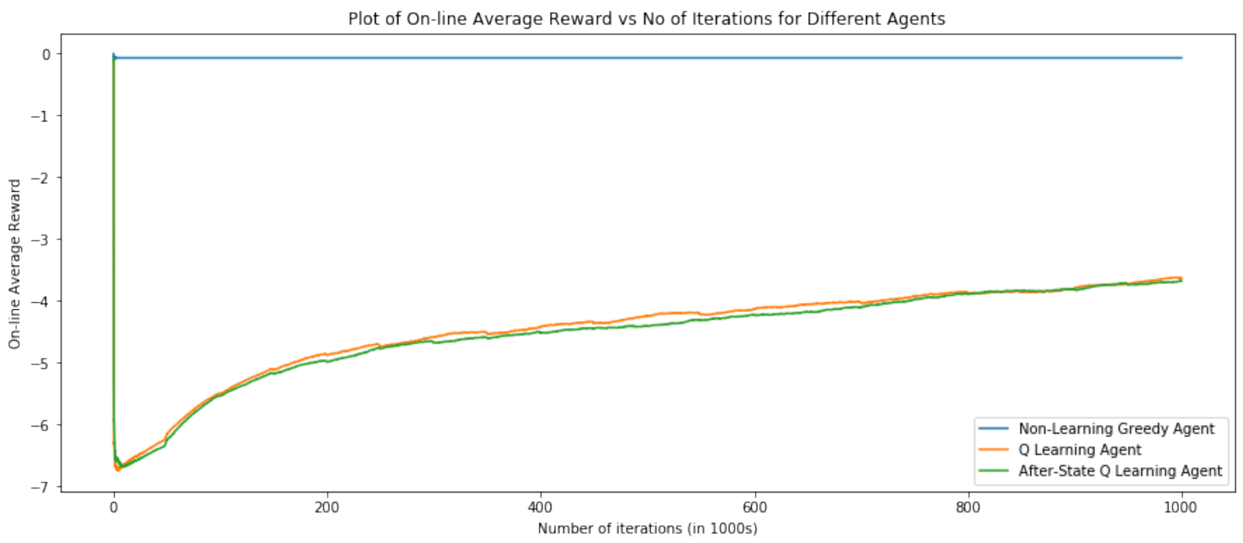

Figure 4: Plot of On-line Average Reward vs No.of Time-steps (30 Trials, 10^6 time-steps, 4 locations, 3 shops, 2 trucks, T = 5)

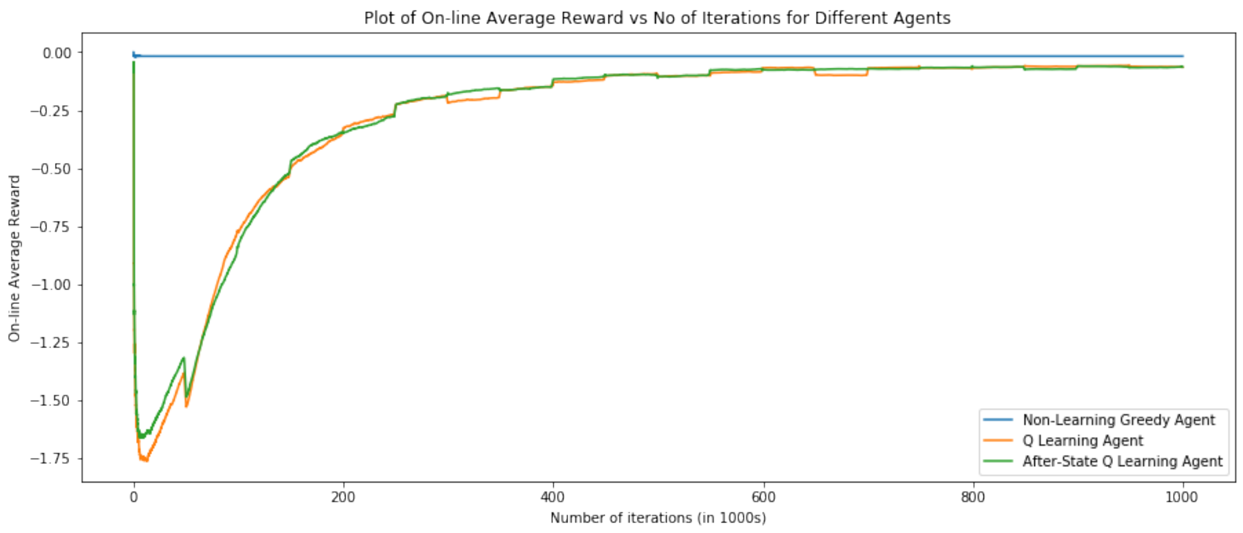

Figure 5: Plot of On-line Average Reward vs No.of Time-steps (30 Trials, 10^6 time-steps, 4 locations, 3 shops, 1 truck, T = 20)

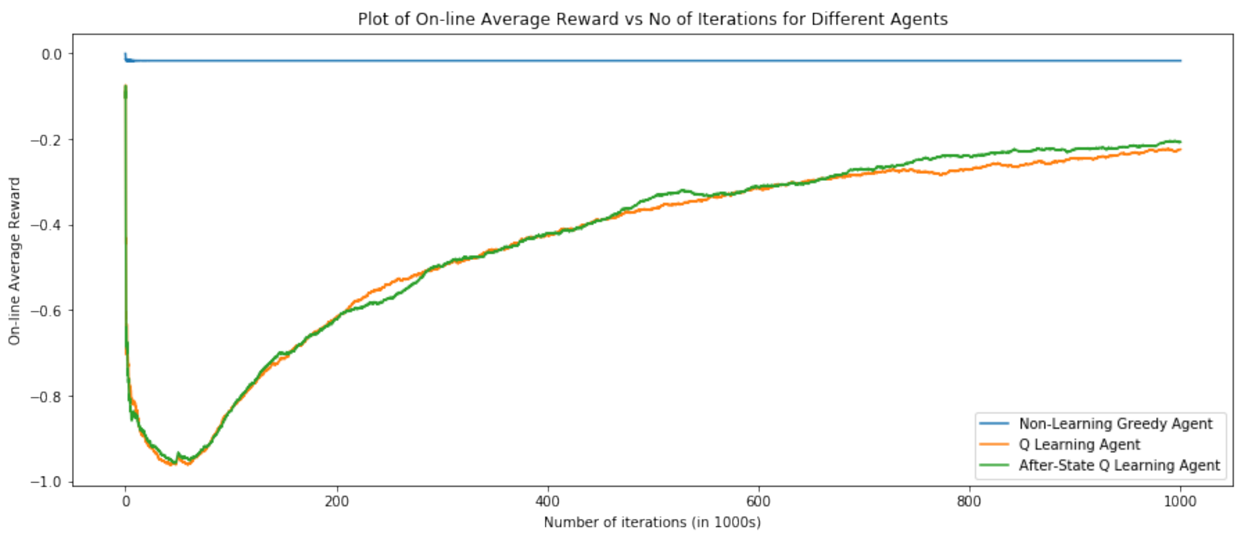

Figure 6: Plot of On-line Average Reward vs No.of Time-steps (30 Trials, 10^6 time-steps, 4 locations, 3 shops, 2 trucks, T = 20)

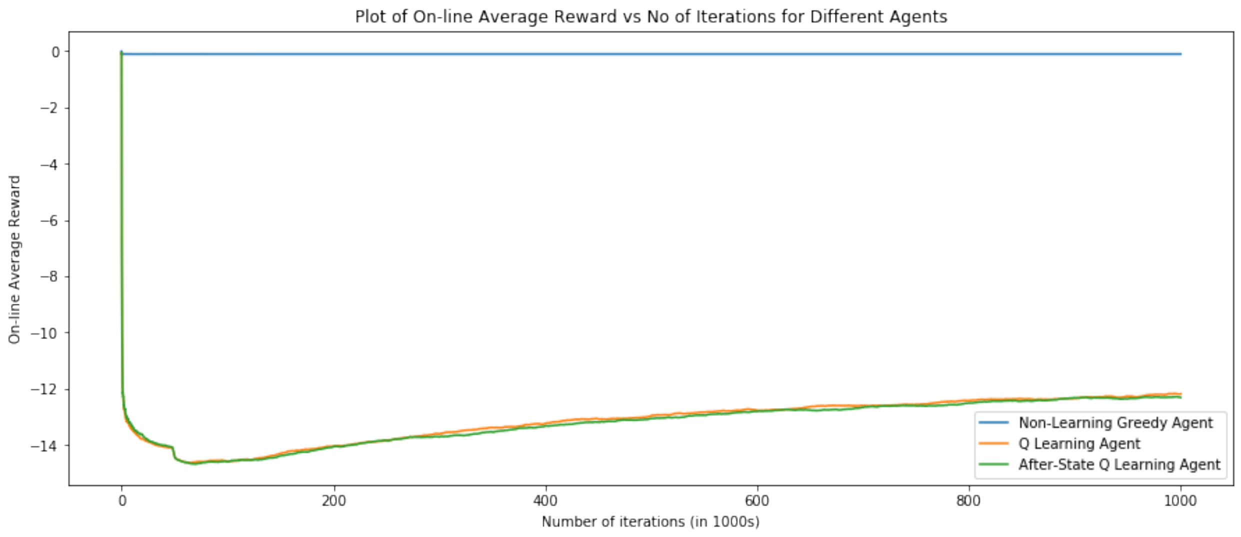

Figure 7: Plot of On-line Average Reward vs No.of Time-steps (30 Trials, 10^6 time-steps, 10 locations, 4 shops, 1 truck, T = 5)

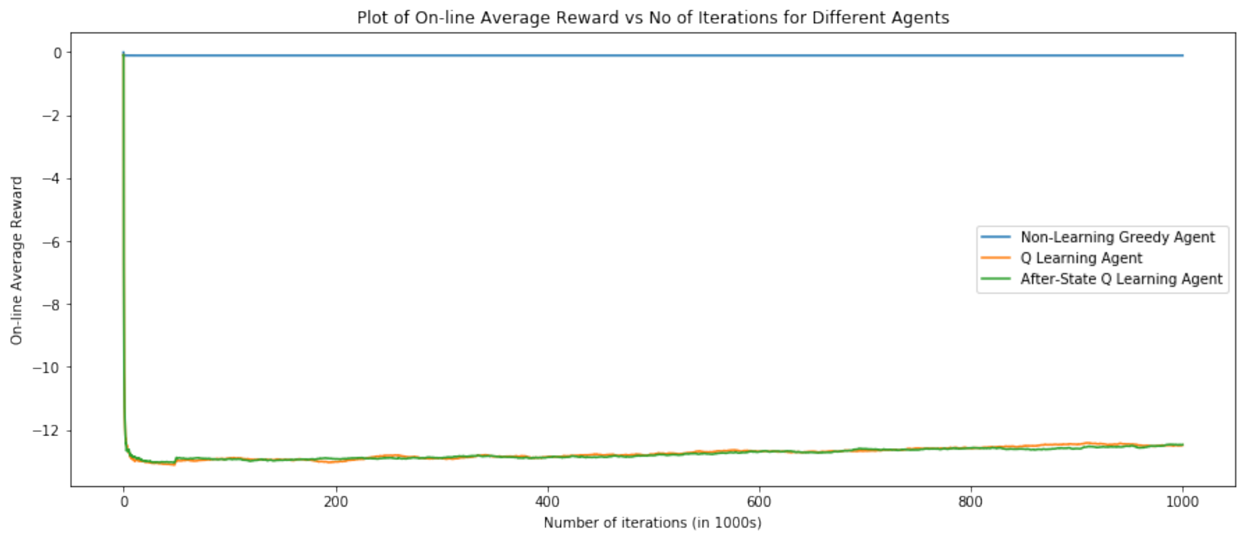

Figure 8: Plot of On-line Average Reward vs No.of Time-steps (30 Trials, 10^6 time-steps, 10 locations, 4 shops, 2 trucks, T = 5)

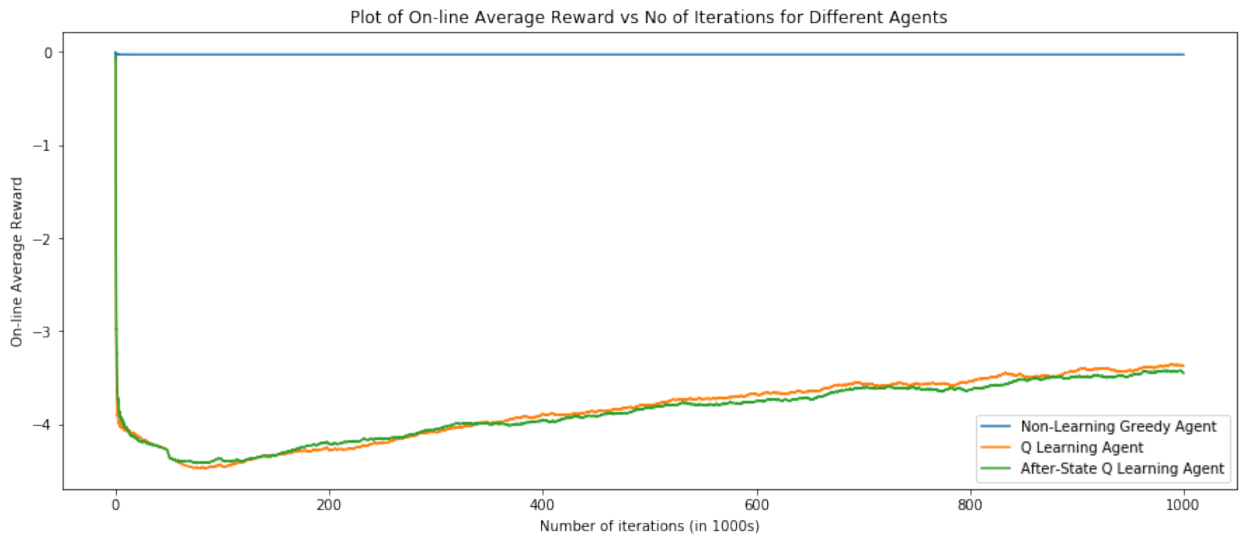

Figure 9: Plot of On-line Average Reward vs No.of Time-steps (30 Trials, 10^6 time-steps, 10 locations, 4 shops, 1 truck, T = 20)

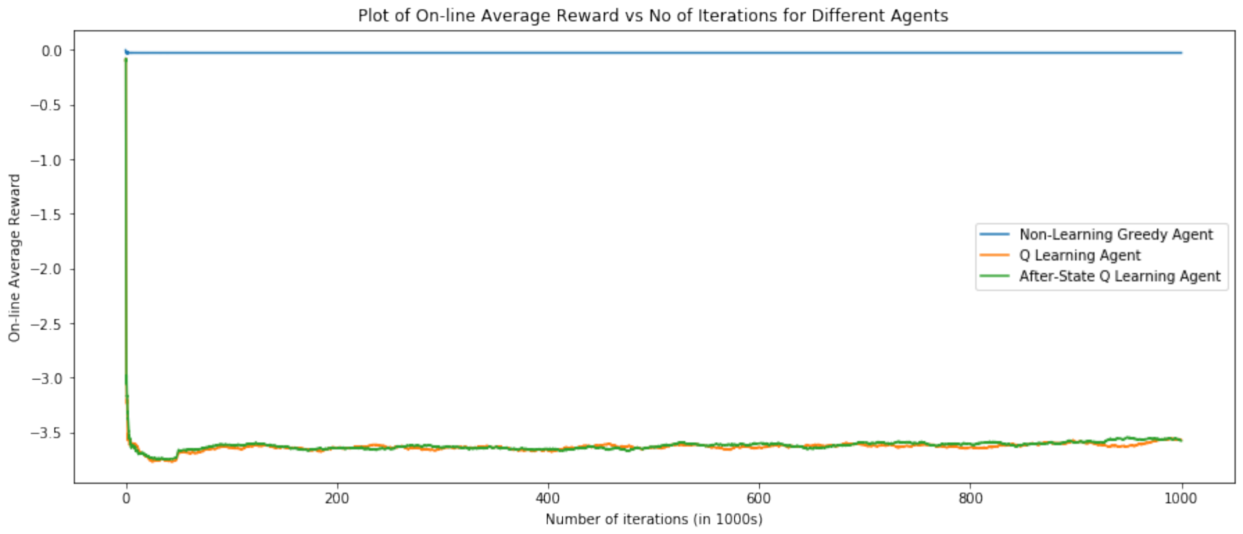

Figure 10: Plot of On-line Average Reward vs No.of Time-steps (30 Trials, 10^6 time-steps, 10 locations, 4 shops, 2 trucks, T = 20)