IE 7295 Semester Group 1 Project: Solving the Vehicle Routing Problem Using Reinforcement Learning

By Karthik Sharma and Hao Wang

1. Introduction

The vehicle routing problem is a classic operations research problem that involves determining the optimal route(s) for a set of vehicles to take while visiting a set of predetermined locations. This problem is similar to the traveling salesperson problem, another operations research problem. However, some critical differences ultimately result in a more complex problem. Traditionally, optimization techniques such as linear programming and metaheuristic algorithms have been applied to solve an optimization problem of this complexity. However, they are limited since the solution is not dynamic and will no longer be optimal if the problem's environment changes significantly. This project aims to evaluate how specific reinforcement learning algorithms will perform when finding an optimal solution to this problem. In theory, reinforcement learning algorithms seem promising for the vehicle routing problem because they are more dynamic and should handle a changing environment better than a linear program or metaheuristic algorithm. The group has decided to use the reinforcement learning environment titled "VehicleRouting-v0" from or_gym[1] to solve this problem. It will use the q learning, dyna-q, and actor-critic reinforcement learning algorithms to find an optimal policy. In this particular version of the vehicle routing problem, the environment takes place in a grid world with a predetermined number of restaurants and delivery driver that must deliver orders from the restaurant to a customer before returning to the restaurant for another order.

2. Problem Statement

The group's problem with this project is which combination of reinforcement learning algorithms and learning settings will produce the optimal policy for a given iteration of the vehicle routing problem. The Dynamic Vehicle Routing Problem is a simulation of a driver working with a food delivery app in a city, accepting and delivering orders with specific delivery values, restaurants, and locations within a time limit. The city is represented as a grid with different zones having varying order creation and value statistics. The driver's vehicle has a finite capacity, and he receives penalties for time and distance spent during travel but rewards for accepting and delivering orders. New orders are created at each time step with a fixed probability unique to each zone[2]. For example, Figure 1 shows a 5*5 grid world with the location of one restaurant (R), driver (D), and delivery location (C).

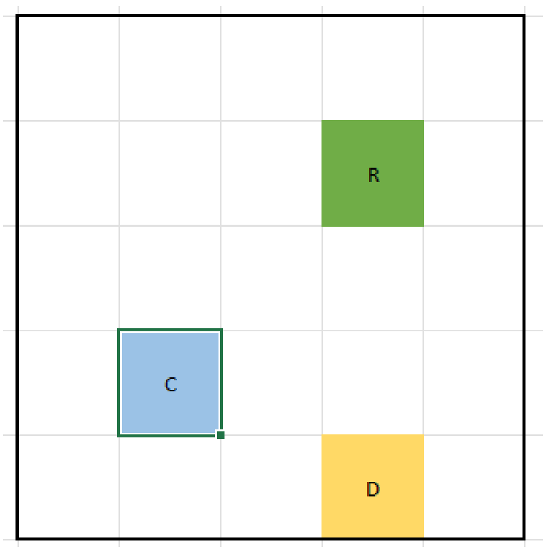


Figure 2-1: *Sample 5x5 Grid Environment*

As mentioned earlier, the three algorithms the group will use in this project are q-learning, dyna-q, and actor-critic. The group will evaluate how these three different algorithms perform on the default vehicle routing environment before moving on to more complex iterations of this problem. For this project, the group decided that the default environment would take place in a 10 x 10 grid, with two driver’s capacity and five restaurants. For making different iterations of the vehicle routing problem, the group will manipulate the three listed features (grid size, number of driver’s capacities, and number of restaurants). For each iteration, one of the three factors will change, and the results will then be recorded to isolate each variable change and see its outcome on each model’s performance. The group will also create different models of the three algorithms by changing each model's epsilon, gamma, and alpha values. For each of the three models, these three variables affect the rate at which the model learns and how much the model prioritizes exploring its environment over exploiting a path that has the best immediate rewards. For this set of experiments, the group decided to use the default setting of the environment. That way, evaluating which model performs best would be more accessible when one of the three parameters is changed.

3. Problem Formulation

This paper focuses on VRP with features of restaurant delivery constraints added. A vehicle with two capacities is responsible for delivering food to destinated nodes. The vehicle could pick up and deliver food from different restaurants during a route. For instance, the vehicle could pick up food from Restaurant A, and during the delivery route, it could pick up another food from Restaurant B and add the second destination to the route. The long route would reduce the rewards, and the key point is to make the right decision for optimal rewards for each route.

4. The Default Environment Setting

This study leverages the VRP environment introduced by or-gym. The default environment is modified based on the following assumptions:

- The city/ delivery zone area is set into a grid to help identify the pickup
- and drop-off locations. For this experiment, the ‘grid size’ is 10 by 10.
- The ‘maximum number of orders’ a restaurant can have is set to 10.
- The ‘order probability’ is set to 50%, indicating the likelihood of receiving an order for pick up at the restaurant.
- Number of restaurants is set to 5.
- The ‘vehicle capacity’ is set to 2, which means the driver can carry 2 orders at a time of travel

5. MDP formulation

5.1 Rewards

We initialize the value to a large positive number to maximize the agent's rewards and assign a small negative value to each move. This incentivizes the agent to pick up the order quickly and take the minimum number of actions necessary to maximize the rewards. The rewards are defined as follows:

- ‘Penalty per move’ is set to -0.2
- ‘Order_promise’ is set to 60, indicating that the driver has 60 minutes for one delivery. If the time limit is reached, the order is identified as missed (time is curved in the environment based on how many states the driver passed)
- ‘Order_miss_penalty’ is set to -50. The driver would receive -50 on the reward for each missed order the driver has.

5.2 Actions

The action space of the agent is the moving direction: up, down, left, and right. The agent can take any of these actions to make a move in the environment.

5.3 States

After the environment assumptions, we define the MDP states in the next part,it contains the following data in order:

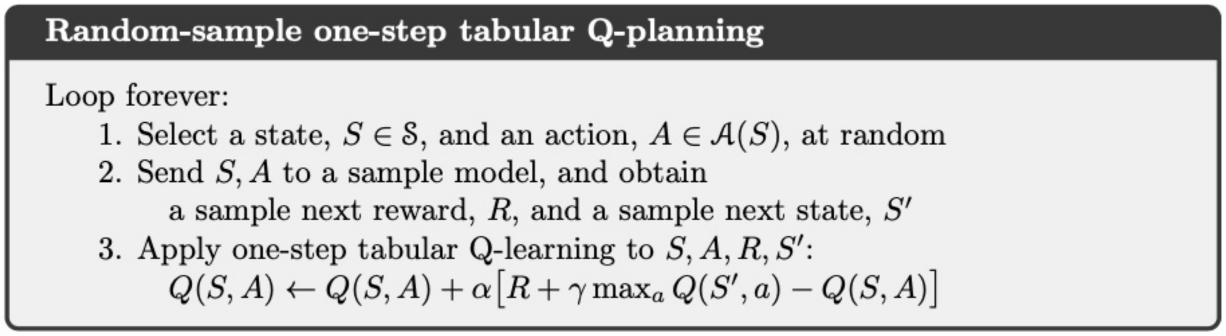
- Restaurant Location (x, y)

- Driver Location (x, y)
- Vehicle Load
- Vehicle Capacity
- Order Status 1 →Acceptable, 2→Accepted, 3→Picked up, 4→ Delivered
- Restaurant ID: The ID of the restaurant the order is from
- Order Drop Location (x, y)
- Time Elapsed: For our case, it is been set to 1

6. Reinforcement Learning Model

6.1 Q-learning model [3]

In this model, the agent maintains a Q-table, which stores each state's expected reward for each action. The agent selects actions based on the highest expected reward in the current state and updates the Q-table after each action based on the difference between the expected and actual reward. Q-learning does not require knowledge of the environment's dynamics or a model of the environment and can handle large state spaces. The algorithm converges to the optimal policy with a sufficiently large number of iterations.



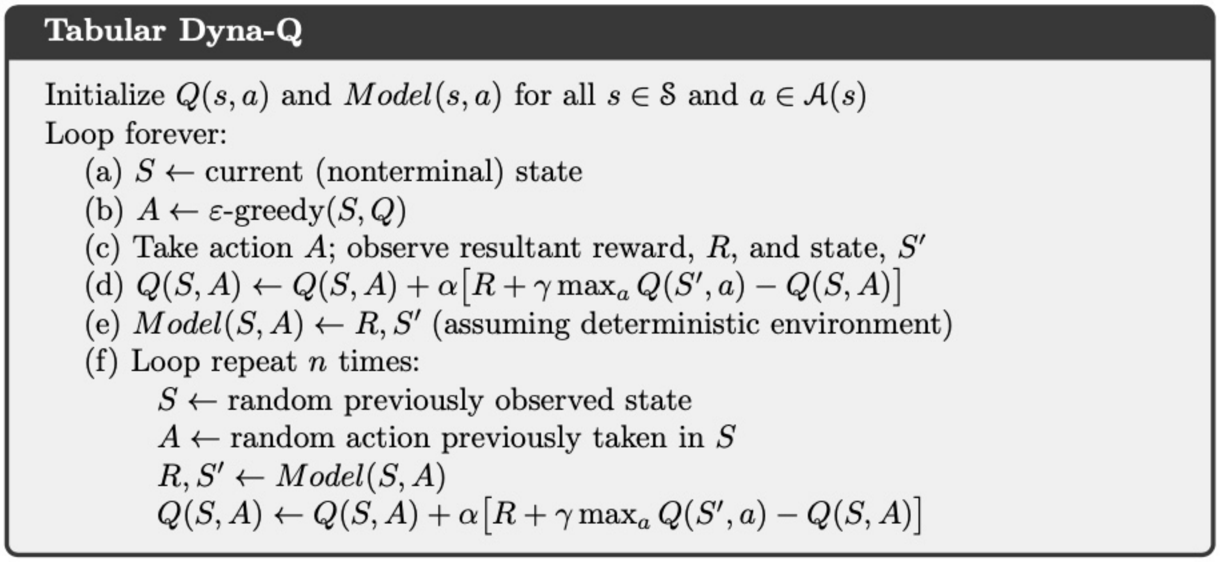
```
Random-sample one-step tabular Q-planning

Loop forever:
    1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
    2. Send $S, A$ to a sample model, and obtain
           a sample next reward, $R$, and a sample next state, $S'$
    3. Apply one-step tabular Q-learning to $S, A, R, S'$:
           $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
```

Figure from Sutton and Barto, 2018

6.2 Dyna Q learning model [4]

In this model, the agent learns a value function using Q-learning and a model of the environment that can be used for planning. Next, the agent uses its current value function to make decisions in the environment and uses the learned model to simulate future states and rewards. The agent then updates its value function based on the simulated experience and the real experience. This allows the agent to learn from its own experience as well as from simulated experience, enabling it to plan ahead and make more informed decisions. As a result, the Dyna-Q model can handle non-stationary environments and can learn more efficiently than other model-based methods.



Tabular Dyna-Q

```
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
    (a) $S \leftarrow$ current (nonterminal) state
    (b) $A \leftarrow \varepsilon$-greedy$(S,Q)$
    (c) Take action $A$; observe resultant reward, $R$, and state, $S'$
    (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
    (f) Loop repeat $n$ times:
            $S \leftarrow$ random previously observed state
            $A \leftarrow$ random action previously taken in $S$
            $R, S' \leftarrow Model(S,A)$
            $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
```

Figure from Sutton and Barto, 2018

6.3 Actor-Critic Model [5]

In this model, the actor learns to choose actions that maximize the expected cumulative reward. In contrast, the critic learns to estimate the value function that evaluates the goodness of the selected actions. The value function is updated based on the difference between the expected and predicted rewards. The actor then adjusts its policy based on estimating the critic's value function. This process is repeated over multiple episodes, gradually improving the actor's policy and the critic's value estimation. The formula for the actor-critic model is as follows:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log(\pi(a|s)) \cdot (Q(s,a) - V(s))$$
$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$$

where θ represents the actor's policy parameters, α represents the learning rate, π(a|s) represents the policy, Q(s, a) represents the estimated state-action value function, V(s) represents the estimated state value function, ∇θ log(π(a|s)) represents the gradient of the log of the policy for the policy parameters, and γ is the discount factor. In this project, the group uses the Python package named stable_baselines3 [6] for the experiment. The Actor-Critic model code is written based on the paper Asynchronous Methods for Deep Reinforcement Learning [7]

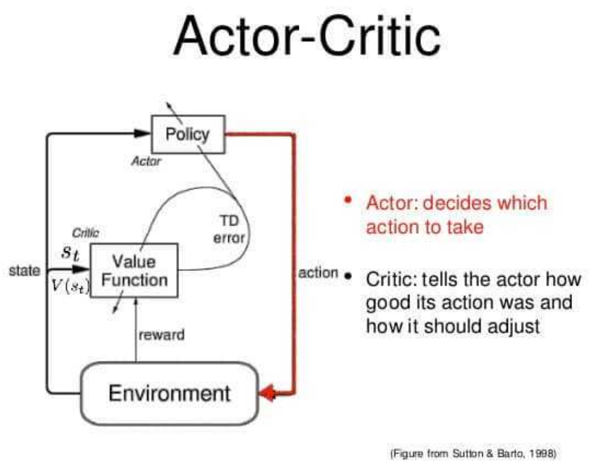
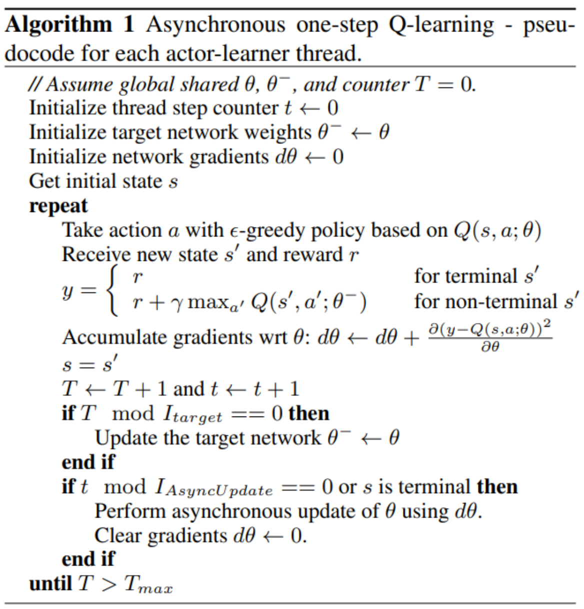

Actor-Critic
Policy
Actor
Critic
TD error
s_t
Value Function
$V(s_t)$
state
reward
action
Environment
Actor: decides which action to take
Critic: tells the actor how good its action was and how it should adjust
(Figure from Sutton & Barto, 1998)

Figure from Sutton and Barto, 1998



Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

```
// Assume global shared $\theta$, $\theta^-$, and counter $T = 0$.
Initialize thread step counter $t \leftarrow 0$
Initialize target network weights $\theta^- \leftarrow \theta$
Initialize network gradients $d\theta \leftarrow 0$
Get initial state $s$
repeat
    Take action $a$ with $\epsilon$-greedy policy based on $Q(s, a; \theta)$
    Receive new state $s'$ and reward $r$
    $y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$
    Accumulate gradients wrt $\theta$: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
    $s = s'$
    $T \leftarrow T + 1$ and $t \leftarrow t + 1$
    if $T \mod I_{target} == 0$ then
        Update the target network $\theta^- \leftarrow \theta$
    end if
    if $t \mod I_{AsyncUpdate} == 0$ or $s$ is terminal then
        Perform asynchronous update of $\theta$ using $d\theta$.
        Clear gradients $d\theta \leftarrow 0$.
    end if
until $T > T_{max}$
```

Figure from Mnih et al., 2016

7. Results

To start, the group decided that there should be a control experiment that can be used to judge other experiment performances off of. For this control experiment, the group decided to use the default model of the environment (10x10 grid world, 5 restaurants, 2 driver's capacities). The figure below shows the results for an environment with 5 restaurants in a 10x10 grid world:

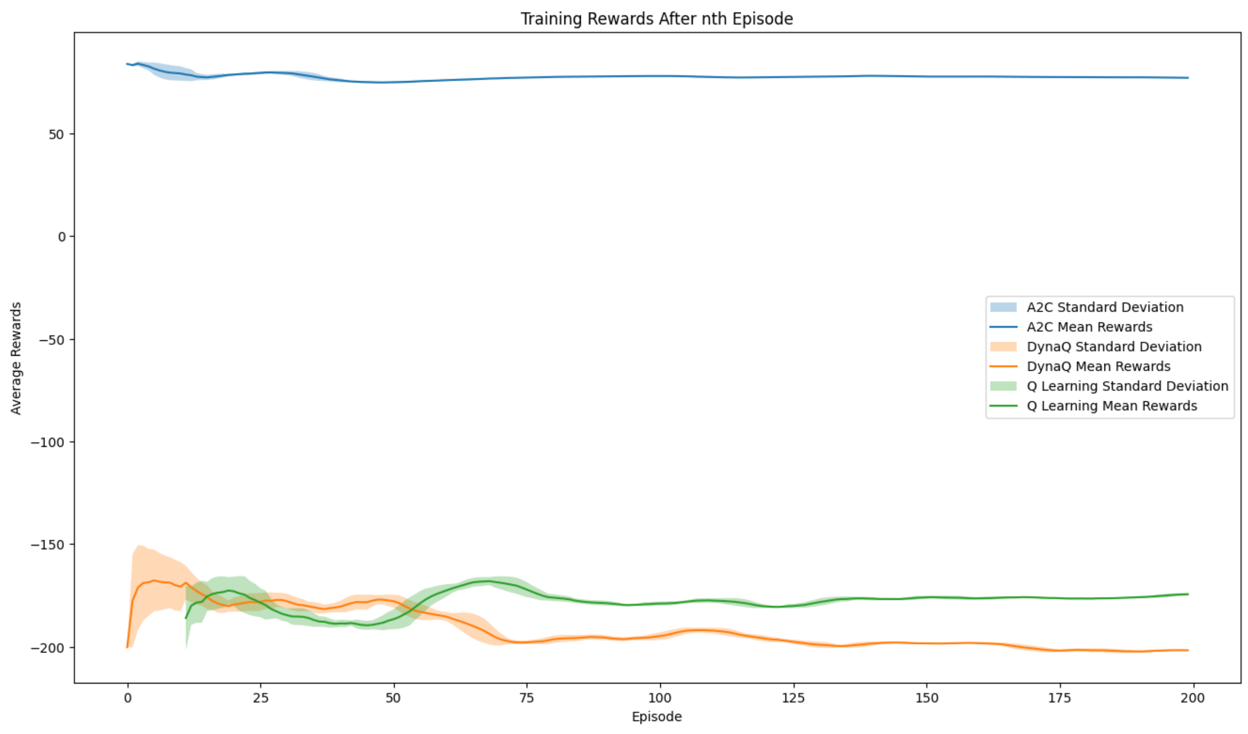


Figure 7-1: *Rewards from the environment with 5 restaurants in a 10x10 grid world.*

For this environment, it is clear that the actor-critic method is the best since its average rewards are positive, and it converges to its optimal policy the fastest out of the three methods fastest. The Q learning and Dyna Q algorithms perform very similarly in this environment, with the q learning mean rewards slightly higher than the dyna q rewards. Both of the algorithms converge after roughly the same number of episodes. After this, the group decided to expand the grid to 15x15 and see how that would impact the three algorithms. The results for that experiment are below:

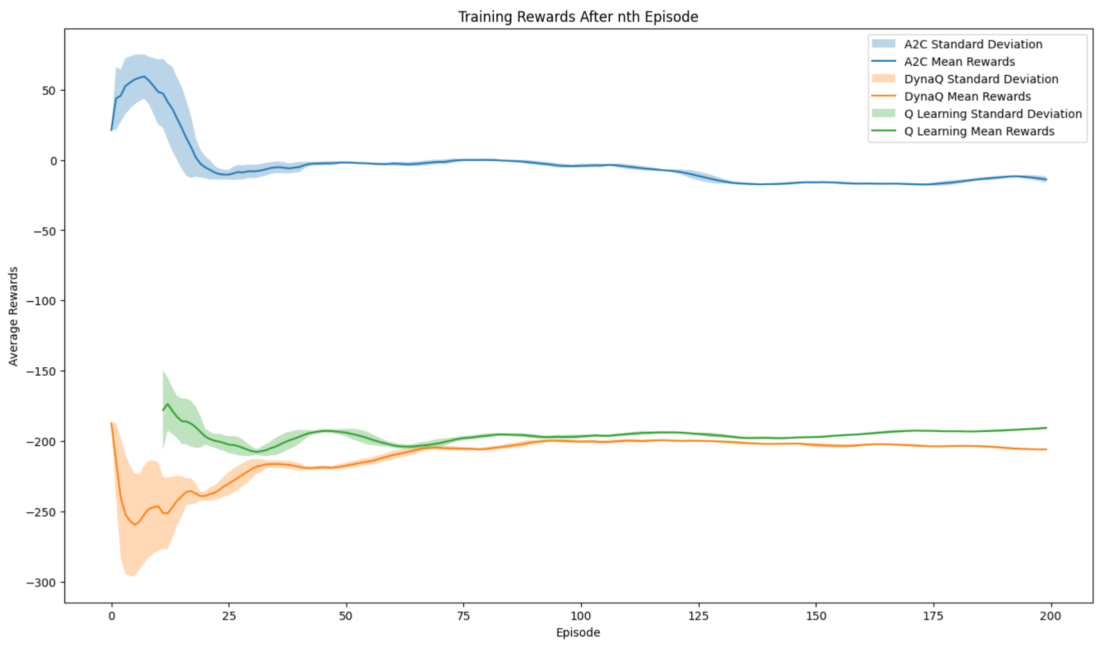


Figure 7-2: *Rewards for 15x15 environment with 5 restaurants*

For this environment, the actor-critic algorithm is still the best since it converges the fastest and has the highest average rewards and converges the fastest. However, it is interesting to note that the average tips for the actor-critic method converge a small number than the convergence for the 10x10 environment. This makes sense since, with a larger environment, the driver's capacities will have to travel more distance to reach their end customers, thus incurring a larger negative reward. This phenomenon can also be seen with the q and dyna q algorithms as their average rewards are now smaller.

Additionally, the difference between the two algorithms' average rewards is smaller than before, suggesting little difference between the optimal policy found by the q learning algorithm and the dyna q algorithm. Next, the group decided to see how increasing the number of restaurants would change the algorithm's performance. Below is the rewards graph for an environment that is in a 10x10 grid with 10 restaurants:

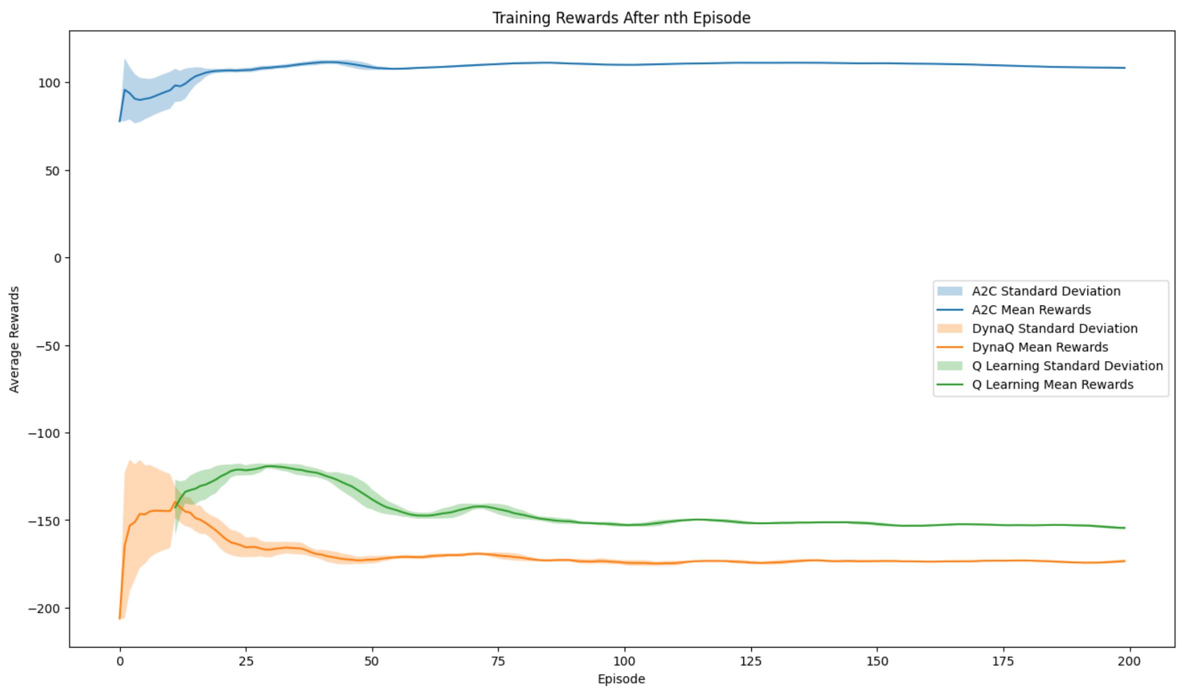


Figure 7-3: *Average Rewards for a 10x10 grid environment with 10 restaurants*

For this environment, it is once again clear that the actor-critic algorithm is the best since it has the highest average rewards out of the three algorithms. An interesting phenomenon to note in this environment is that all three algorithms' average rewards are higher. This makes sense since there are more restaurants and orders to complete, which should result in more rewards. There is no change in performance for the q learning and dyna q algorithms with respect to each other when comparing this environment to the control experiment. The next experiment the group tried was to see what would happen if the number of driver’s capacity increased from 2 to 5. The results are shown below:

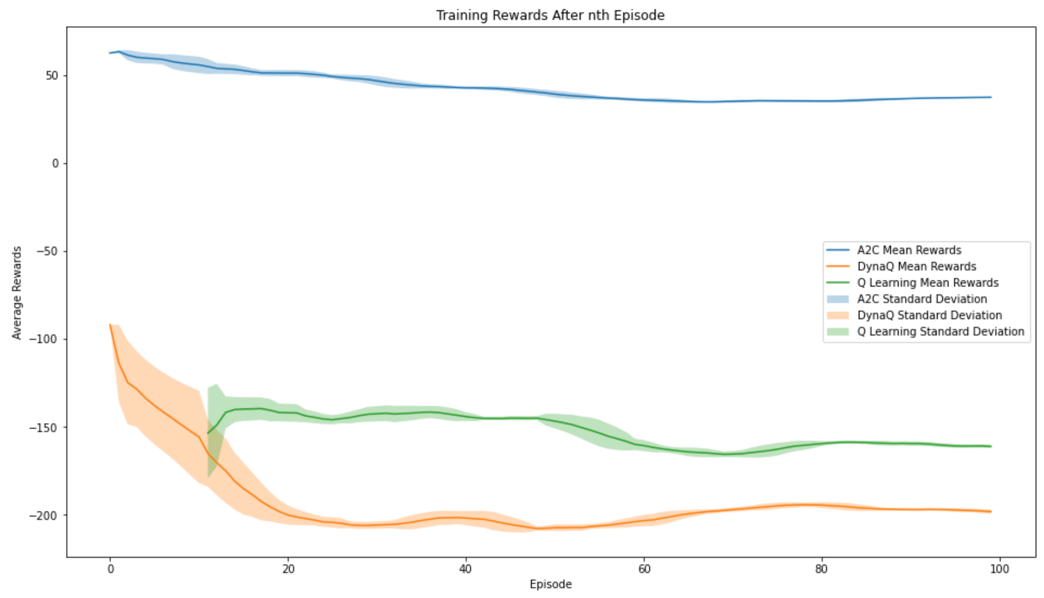


Figure 7-4: *Average Rewards for a 10x10 grid environment with 5 driver capacity*

As seen in the graph, the actor-critic model once again is the best since it has the highest average rewards out of the three algorithms. The overall behavior for the algorithms in this environment is very similar to the how the algorithms performed in the control experiment environment. The next factor that the group experimented with was the learning rate for each algorithm. For the q learning and dyna q algorithms, the team changed the learning rate (alpha) from 0.1 to 0.4 and changed the learning rate for the actor-critic algorithm from 0.0005 to 0.002. The three algorithms' performances can be seen below:

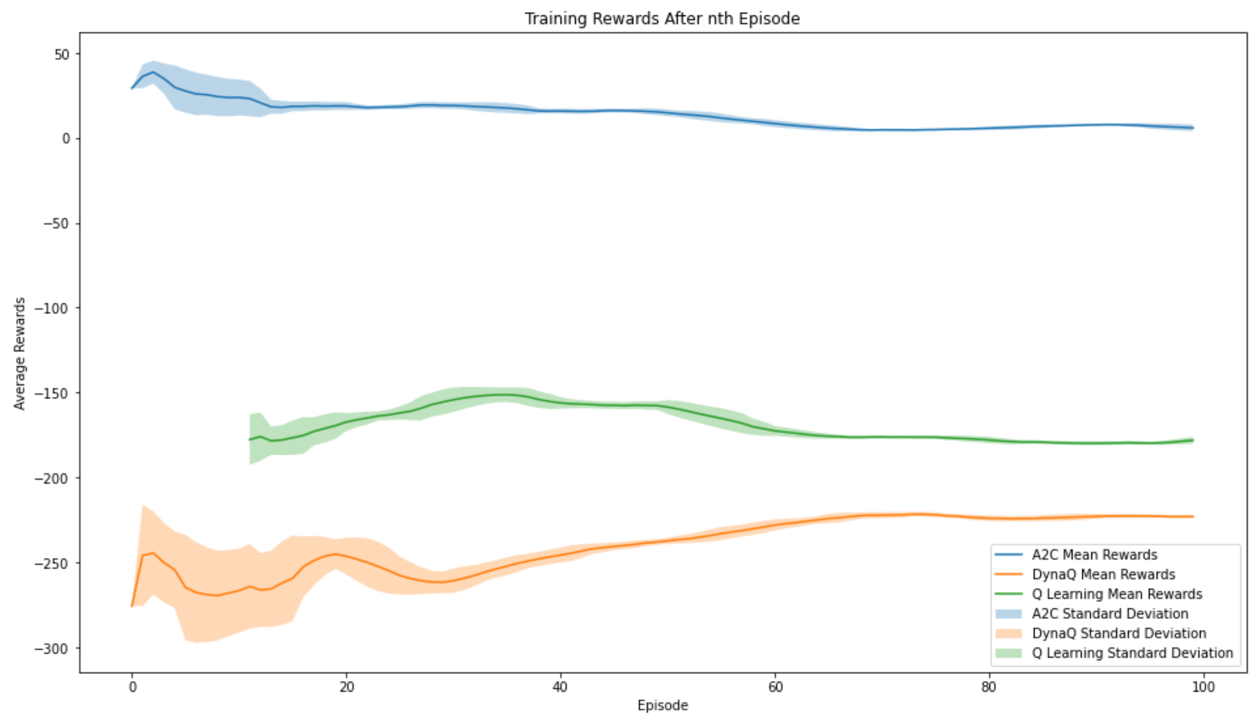


Figure 7-5: *Rewards for Models With Larger Learning Rates*

The actor-critic method is still the best, given this graph. The q-learning algorithm starts with a less negative average reward than the control for this project. However, the optimal policy for both it and dyna q algorithm reach is still similar to the base case. Next, the group decided to lower the learning rates for the three algorithms to 0.05 for q learning and dyna q and 0.00025 for the actor-critic model. The results are below:

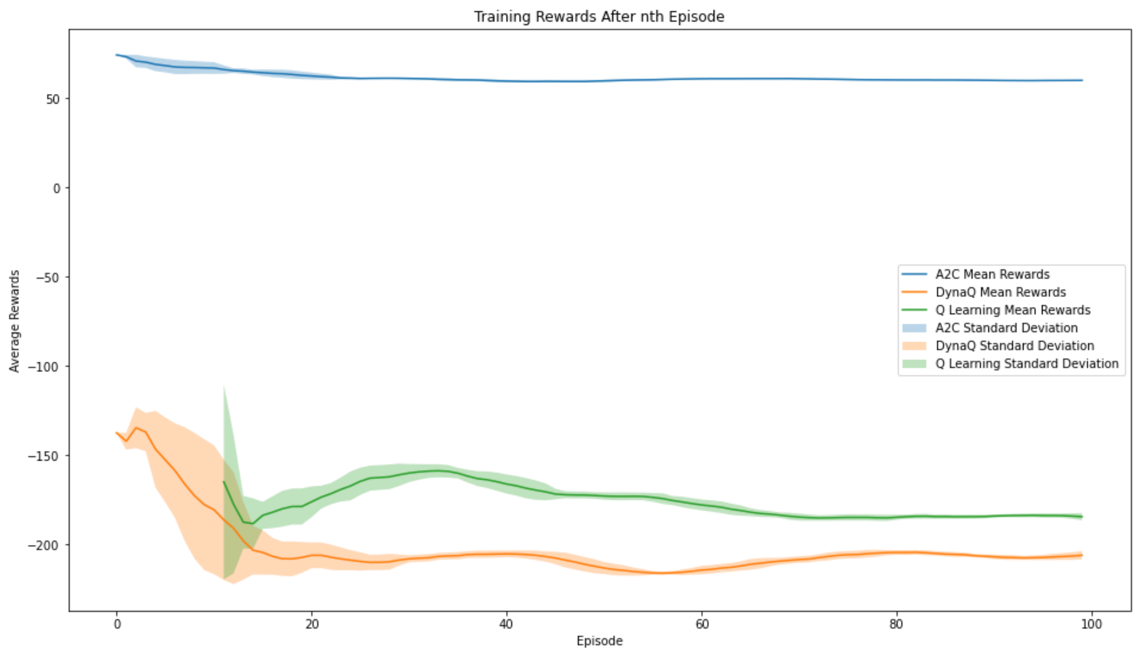


Figure 7-6: *Rewards With Lower Learning Rates*

The actor-critic method is the best again, but an interesting phenomenon occurs here. For all three models, the model's average reward starts out high and then declines slowly over time before converging to the optimal policy. The end policies the models converged to are very similar to those chosen by the baseline models if we base the evaluation on average rewards received. The next experiment that the group tried was lowering the gamma value to 0.85 for all three models. The results are shown below:

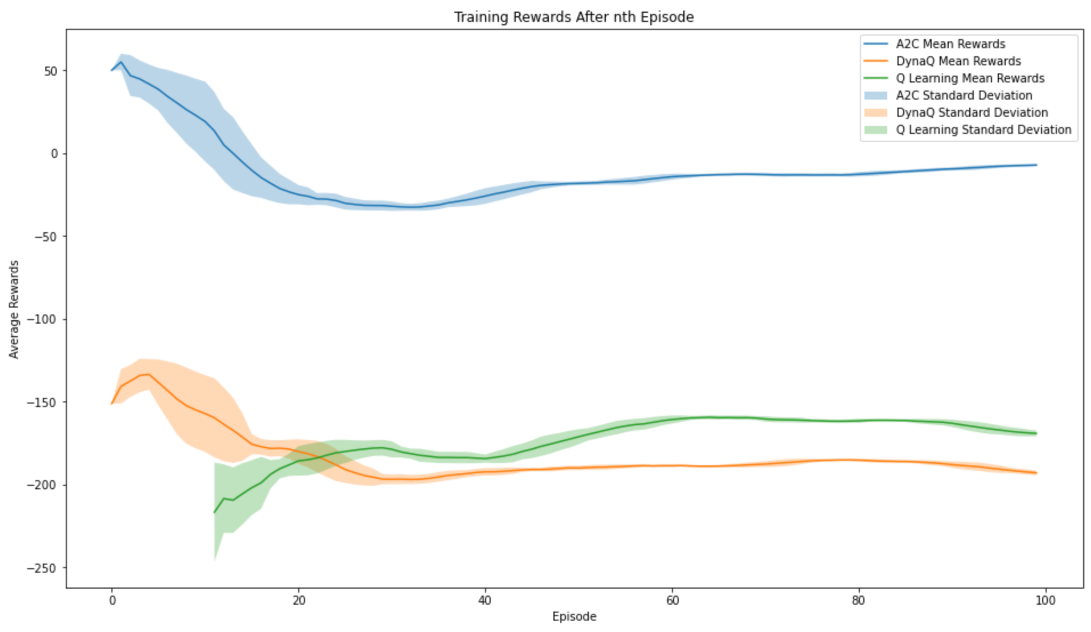


Figure 7-7: *Rewards for Models With Lower Gamma Value*

Although the actor-critic model is still the best with these parameter settings, it performed worse than the baseline setting since its average rewards are lower than the average rewards in its baseline model. The gamma change did not affect the q learning or dyna q algorithms. The last change was to change the epsilon value in the q and dyna q models from 0.1 to 0.4.

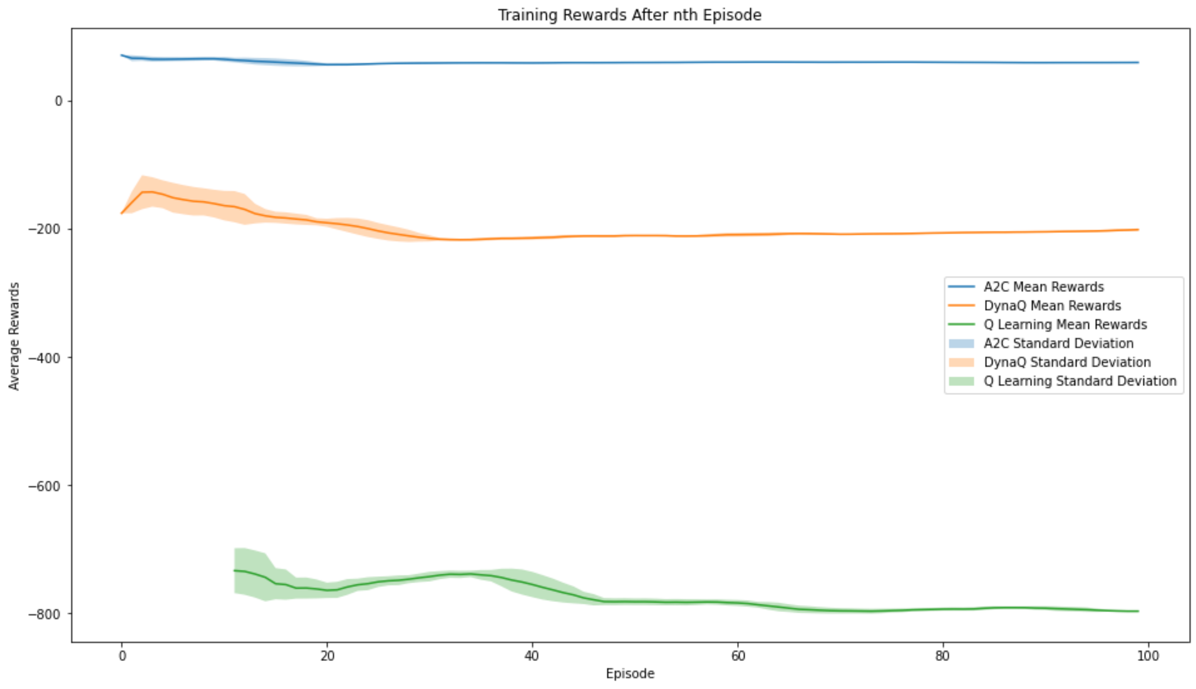


Figure7-8: *Rewards for Models With Larger Epsilon Value*

The new epsilon value has greatly hurt the q-learning model since the average rewards are much lower than they were in the baseline model. However, the increased exploration did not affect the dyna q model as much as its average rewards are largely the same as they were in the baseline case.

8. Conclusion

In conclusion, the group conducted a series of experiments to evaluate the performance of three reinforcement learning algorithms: q learning, dyna q, and actor-critic, in a simulated food delivery task. The experiments varied the environment size, number of restaurants, number of driver's capacities, learning rate, gamma value, and epsilon value. Overall, the actor-critic algorithm consistently outperformed the other two algorithms regarding convergence speed and average rewards. The experiments also revealed that the environment size and the number of restaurants and driver's capacity impacted the algorithms' performances, with larger environments and more restaurants and driver's capacity resulting in lower average rewards. The experiments on the learning rate, gamma value, and epsilon value showed that changing these hyperparameters can significantly affect the algorithms' performances and, in some cases, hurt their performance. These findings demonstrate the importance of carefully tuning hyperparameters in reinforcement learning algorithms to achieve optimal performance.

References

[1] Hubbs, Christian D., et al. "OR-Gym: A Reinforcement Learning Library for Operations Research Problems." *ArXiv.org*, 17 Oct. 2020, https://arxiv.org/abs/2008.06319.

[2] Balaji, Bharathan, et al. "ORL: Reinforcement Learning Benchmarks for Online Stochastic Optimization Problems." *ArXiv.org*, 1 Dec. 2019, https://arxiv.org/abs/1911.10641.

[3] [4] [5] Sutton, Richard S., et al. *Reinforcement Learning: An Introduction*. MIT Press Ltd, 2018.

[6] Mnih, Volodymyr, et al. "Asynchronous Methods for Deep Reinforcement Learning." *ArXiv.org*, 16 June 2016, https://arxiv.org/abs/1602.01783.