Model Free Reinforcement Learning for On-Demand Delivery Optimization

Karthik Pythireddi
School of Engineering
Stanford University
Stanford, CA, USA
karthik9@stanford.edu

Siya Goel
School of Engineering
Stanford University
Stanford, CA, USA
siyagoel@stanford.edu

Arman Sharma
School of Engineering
Stanford University
Stanford, CA, USA
arman02@stanford.edu

***Abstract*—Doordash is a food delivery service that is used globally, transforming and enhancing the food delivery process. This paper discusses the application of Q-learning to an on-demand food delivery scenario in order to achieve an optimized delivery time. We modeled our problem as a Markov Decision Process (MDP) where the environment is comprised of grid cells and the goal of the dasher is to reach the customer cell, the reward, in the least amount of steps. We coupled the Q-learning model with ε-Greedy exploration to have a balance in both exploration and exploitation and also to aid in the convergence of the optimal policy. We found that the agent that uses Q-learning reached the customer in fewer steps compared to the random agent, especially for lower epsilon (ε) values, as there was less exploration. We also found that increasing the gamma (γ) value within the Q-learning algorithm can lead to faster delivery. In the future, more experimentation can be done to see how models (like Q-learning and deep Q-learning) perform when more uncertainty, such as the introduction of dashers and obstacles, is added to the environment.**

***Index Terms*—Q-learning, ε-Greedy, MDP, Delivery**

I. Introduction

Doordash is a leading on-demand food delivery service that connects customers with a diverse range of local restaurants. As a pioneer in the gig economy, Doordash has transformed the way people access their favorite meals, offering remarkable convenience and efficiency. Behind the scenes, Doordash relies on sophisticated algorithms to optimize various aspects of its operations. Its comparative advantage as a service in the post-COVID economy is its reliability, which a huge amount of algorithmic expertise supports. According to Li and Wang [1], these algorithms play a crucial role in enhancing the overall user experience, from personalized restaurant recommendations based on individual preferences to dynamic pricing strategies that balance supply and demand. However, most core to Doordash's operations and survival as a company are those algorithms for route optimization, ensuring timely and efficient deliveries by considering factors such as traffic patterns and delivery distances. In essence, Doordash's success is not only grounded in its seamless user interface, but also in the intelligent algorithms working behind the scenes to streamline and enhance the food delivery process.

The route optimization algorithms must take into account a myriad of factors to ensure timely and seamless food deliveries. These factors include a range of dynamic variables like traffic and weather conditions, some of which are easier to predict than others. At the core; however, are variables like the distance between the customer and the pickup location. Minimizing the delivery time for customer satisfaction relies heavily on such fundamental variables.

As such, our goal is to minimize the delivery time of a dasher agent in a model environment. Specifically, we model a geographic area as a grid space with coordinates and employ Q-learning with an exploration strategy to find the optimal dasher policy.

II. Related Work

Reinforcement learning provides a dynamic approach to solving complex problems by allowing agents to learn from interactions with their environment. Unlike traditional supervised learning, reinforcement learning focuses on making sequential decisions to maximize total rewards. Reinforcement learning is uniquely effective in scenarios where preprogramming is challenging and where the environment is always changing, such as in robotics, finance, and gaming. Learning from trial and error and adapting strategy based on rewards and/or penalties, reinforcement learning excels in addressing problems characterized by complexity, uncertainty, and the need for adaptive strategies.

Q-learning has by far been one of the most commonly used reinforcement learning methods. Chen, Ulmer, and Thomas [2] used Q-learning for same-day delivery with fleets of drones and vehicles, where the method learns the value of assigning a new customer to either drones or vehicles as well as the option to not offer the service at all. They create a Euclidean plane to represent the area in which deliverers and customers are located, and at every decision point, they update a "planned route", which represents the tentative routes vehicles and drones follow until the next decision update takes place.

Silva, Pedroso, and Viana [3] apply deep reinforcement learning to solve the problem of crowdshipping, which is when a company allows the services of additional, part-time drivers who use their own vehicles in return for a small fee. They follow a two-staged decision model where the first decision is the order in which all customers will be delivered and the second decision defines routes that follow the first-stage decision ordering.

Revadekar, Soni, and Nimkar [4] use Q-learning for pharmacy delivery optimization. They use the PharmaQuick ecosystem, which provides an e-commerce service for ordering medications, and incorporate distance and time parameters into their Q-learning model. With a 2D 10-by-10 map, they run thousands of episodes that explore new route patterns to find the optimal delivery routes. Although this paper's chosen variable to optimize (route patterns) was closest to the dasher problem we examined, we have not found any published literature on solving the food delivery problem using a Q-learning approach.

Specifically, in the past, there have been many other machine-learning approaches used to model food delivery times. Gao et al. [5] used deep learning to accurately estimate a driver's delivery time and route using a model called FD-RTP. The model predicts the probability that the delivery driver visits a particular spot. However, Adak, Pradhan, and Shukla [6] state that deep learning models lack explainability and are hard to implement practically. As a result, they are not used often in optimizing food delivery systems. Park and Oh [7] stated that Q-learning could be a potential solution to this problem as it provides explainable recommendations of paths a dasher could take to find the customer.

III. Methods

We used a model-free approach to solve the problem, as our agent interacts with the environment shown in Fig (1) for every action. As we were not sure of the transition probabilities and reward models, using a model-free method like Q-learning felt applicable to our problem statement.

Q-learning involves the learning of the action value function *Q(s,a)*. The incremental update rule for the actions value function is given by:

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

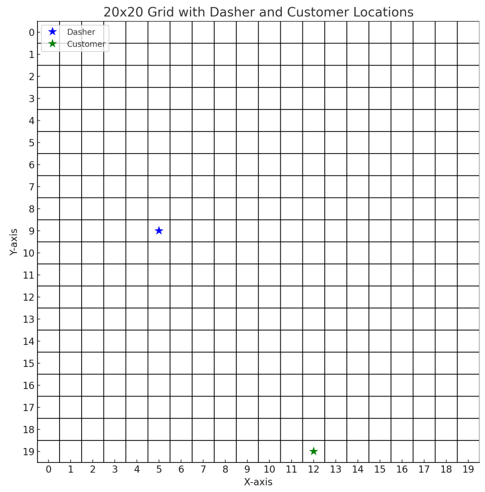


Fig. 1: Grid Cell Approach and Environment Used for Modeling

In $eq[1]$, α is the learning rate and the range of values is given by $0 \leq \alpha \leq 1$. The higher the α the lesser the focus on the older information. Instead, the model tries to learn from the new information. Thus, the learning rate is vital for the Q-learning algorithm as it controls the weight that needs to be given between older data and the newly acquired data. There's another variable called discount factor ($0 \leq \gamma \leq 1$) which can be varied depending on how much weight needs to be given for future rewards. If the value is closer to zero then more priority will be given to the current rewards. As the value increases, the future rewards are considered more.

The $\max_{a'} Q(s',a')$ in $eq[1]$ calculates the maximum q-value for the next state s' for all possible actions a'. After taking an action a in the state s we achieve a reward r and the new state s'. As a result, the action value function $Q(s,a)$ gets updated. The update includes both the past and the current information i.e. the summation of the reward with the discounted maximum future values subtracted from the past values.

We used an undirected exploration strategy called ε-Greedy exploration in conjunction with our Q-learning algorithm to balance the trade-off between exploration and exploitation. Here, exploration refers to finding newer areas of the environment as larger values of epsilon (ε) lead to more randomness. Specifically, the value of the epsilon (ε) defines the amount of randomness while choosing an action for exploration. Exploitation is making use of the knowledge learned using the exploration strategy. Thus, the 1-ε chooses the best action by making use of its current q-value estimates, describing the exploitation strategy.

IV. Model

Our current problem involves making sequential decisions. The problem consists of a customer placing an order and it being delivered by the dasher in an optimal amount of time. This problem can be modeled into a *Markov decision process (MDP)*, as shown in Fig (2), which is the standard model for the sequential decision problems.

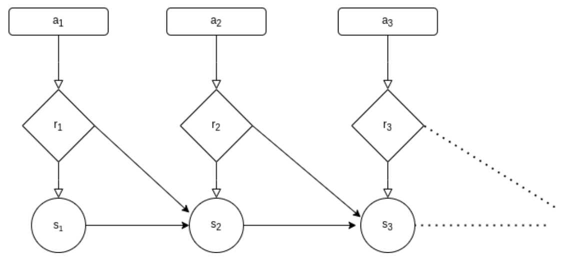


Fig. 2: Markov Decision Process

Our (s, a, r, s') for the MDP are defined as below:

State (s)	The current position of the dasher in the grid world.
Action (a)	The act of a dasher moving from one cell to another.

Reward (r)	The value which the dasher receives after executing an action.
Next State (s')	The state that the agent is in after executing an action.

We modeled our environment into a grid world, where the customer location is fixed and the dasher tries to find the optimal path to the customer. The environment was simulated in such a way that the interaction between the dasher and customer mimics the Doordash model. Some basic variables that were introduced in the environment were the grid size, which is kept constant at $(20, 20)$, the dasher location, and the customer location. The customer location had a maximum reward of $+10$ and other locations in the grid had a reward of -1. The dasher location was chosen randomly but the customer location was fixed in the simulation.

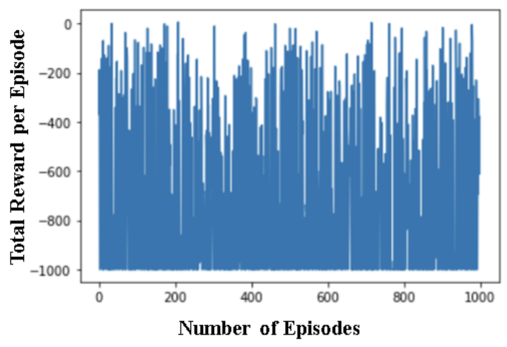
The interaction of the agent, or the dasher, with its environments starts at the decision stage $(t = 0)$ with the observation of the current state s_0. Note that there is no reward in this initial stage. The agent then chooses an action to execute at the decision stage $(t = 1)$. The environment responds by changing its state to s_1 and returning the reward r_1. The agent has four possible actions in each state (up, down, left, and right) which are chosen randomly. If the agent gets blocked in a particular direction, then it remains in the same grid square. The grid square with the customer location is the terminal state. If the dasher finds itself at the customer's location, the episode ends. Then, a new episode begins with the dasher at a random initial state. Two different agents were created:

- We created a *random agent* that chooses actions randomly within the environment. This agent moves randomly and does not learn from its actions. This gives a baseline performance to compare against the Q-learning agent. The random agent is run for 1000 episodes and each episode is run for a maximum of 1000 steps or until the dasher finds the customer location.
- We created an agent that uses Q-learning. The *Q-learning agent* was run for 5000 episodes and each episode was run for a maximum of 1000 steps or until the dasher finds the customer location. We used initial q-values of 0 and the stochastic parameters for the ε-greedy policy function were initially set to epsilon $(\varepsilon) = 0.05$, discount factor $(\gamma) = 0.9$, and learning rate $(\alpha) = 0.1$. We then experimented with the epsilon (ε) values to introduce the ε-Greedy approach to analyze the effect of different amounts of exploration and exploitation. Additionally, the gamma (γ) values were varied to see the effect of future rewards in optimizing delivery time.

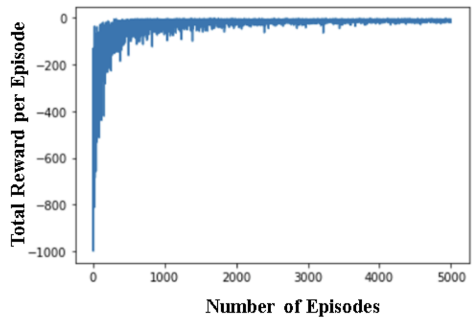
The total reward is obtained by the two different agents throughout the episodes. Additionally, the learning curve is obtained by running enough episodes on the Q-learning agent so that the agent converges to a near-optimal policy.

V. Results

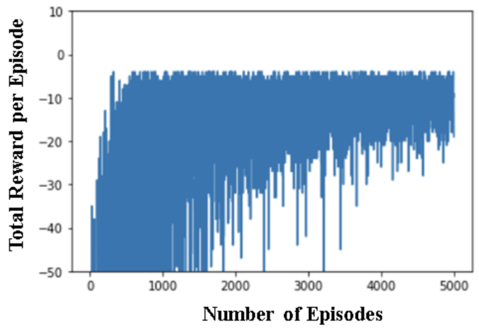
From the plots in Fig (3), we can see the difference in the reward per episode between the random agent and the



(a) Random Agent's Reward per Episode



(b) Q-learning Agent's Reward per Episode

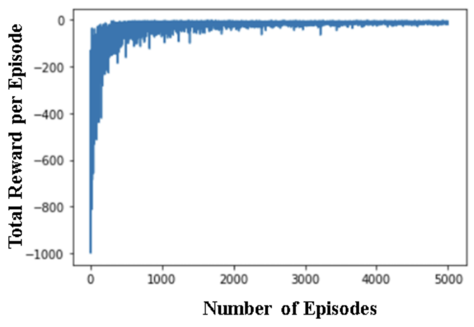


(c) Zoomed in Graph for Q-learning Agent's Reward per Episode

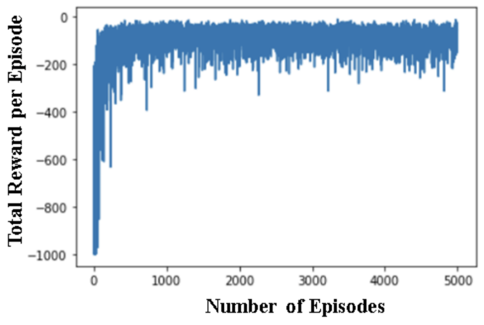
Fig. 3: Comparison of Reward per Episode Between Random and Q-learning Agent

Q-learning agent. The minimum reward in the simulation for any episode is -1000 as the maximum number of steps in each episode considered in the simulation is 1000. The reward of -1000 means that the dasher was not able to reach the customer location in the allocated steps. In Fig (3a), one can see that the random agent does not learn and the performance is not optimal. This is because the reward per episode varies greatly in the range of -1000 to 0.

However, as shown in Fig (3b), the Q-learning agent learns as it converges to a larger reward per episode at around 1000 episodes. When analyzing the zoomed-in graph of the Q-learning agent's results in Fig (3c), we see that the maximum reward from 5000 episodes was -4. This means that it took at least 7 steps for the dasher to reach the customer. Thus, the reward achieved from the Q-learning agent is a significant



(a) Q-learning Agent's Reward per Episode (ε = 0.05)



(b) Q-learning Agent's Reward per Episode (ε = 0.75)

Fig. 4: Comparison of the Q-learning Agent's Reward per Episode at Different Epsilon Values

improvement over the reward obtained from the random agent. This is because an optimal reward is achieved with the Q-learning agent unlike the variability shown in the rewards produced by the random agent.

Fig (4) shows the comparison in the performance of the Q-learning agent using epsilon (ε) = 0.05 and 0.75. Even though the rewards per epsiode for both models converge, the model with the larger value of epsilon (ε), Fig (4b), has more variability in the rewards per episode compared to the model with the lower epsilon (ε) value, Fig (4a). This is because larger values of epsilon (ε) lead to more exploration, and therefore more steps are wasted to get to the customer location. From the action value function $Q(s, a)$, the policy can be obtained using the following equation:

$$\pi(s) = \underset{a}{argmax} Q(s, a) \tag{2}$$

In addition, we looked at the mean number of steps taken to reach the terminal state by a random agent in a simulation of 1000 episodes and a Q-learning agent in a simulation of 5000 episodes. The standard error is calculated as $\frac{\sigma}{\sqrt{n}}$ where n represents the number of episodes. The results between the Q-learning agent and random agent were compared using a variety of different epsilon (ε) values. Additionally, the results of the Q-learning agent were compared to itself by changing gamma (γ) values. Fig (5) shows that changing the epsilon (ε) values did not have any significant effect on the mean number of steps taken by the random agent.

However, the epsilon (ε) values had a different effect on the Q-learning agent. Specifically, as shown in Fig (6), increasing the epsilon (ε) value led to more exploration. Thus, the mean number of steps taken by the dasher to reach the customer increased drastically. In fact, when the epsilon (ε) $= 1$, one can see that the Q-learning agent will behave similarly to the random agent. Additionally, we found that increasing the discount factor, or gamma (γ), reduces the mean number of steps slightly, especially for lower values of epsilon (ε).

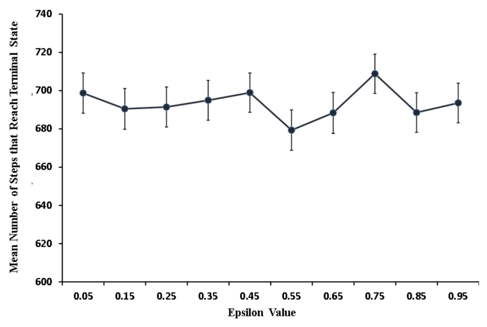


Fig. 5: Mean Number of Steps that Reach Terminal State by the Random Agent for Multiple Epsilon Values $(n = 1000)$

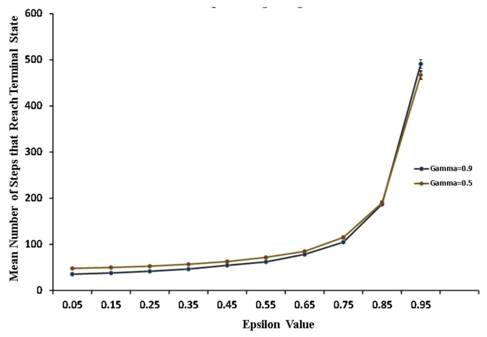


Fig. 6: Mean Number of Steps that Reach Terminal State by the Q-learning Agent for Different Epsilon and Gamma Values $(n = 5000)$

Fig 7 shows the policy for all states in a 20 x 20 grid for the Q-learning agent with an $(\varepsilon) = 0.05$. The purple dot in the figure shows the dasher's location and the blue dot shows the customer's location. Here, one can see the optimized action the dasher takes at each grid location to reach the customer location in the least amount of steps. Specifically, a variety of actions are used so that the dasher can reach the customer in a shorter period of time.

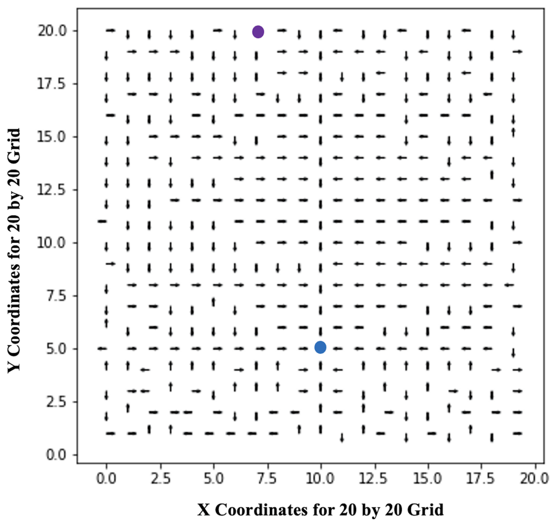


Fig. 7: Q-learning Agent's ($\varepsilon = 0.05$) Policy for all States in a 20 x 20 Grid

VI. Future Work

There are two main improvements we could make to this work. The first would be the use of deep Q-learning instead of the standard Q-learning agent we've employed here. While Q-learning uses a tabular approach to store Q-values for state-action pairs, it can be somewhat limited for complex and high-dimensional state spaces. Deep Q-learning addresses this limitation by employing a neural network to approximate Q-values, allowing it to handle intricate or continuous state representations. Like Bi et al. [8], we could employ deep Q-learning to estimate Q-values for our state-action pairs.

The second would be the incorporation of more variables into the model. Currently, the only variable we are examining is 2-dimensional distance. In reality, there are dynamic variables like peak ordering data, lifestyle, culture and traffic conditions that have to be taken into account. We could introduce stochastic variables to represent such dynamic variables, like that done by Wu and Xu [9], making certain routes more difficult to travel across and further optimize as time goes on.

VII. Conclusion

In this project, we modeled a Doordash food delivery service using Q-learning with ε-Greedy exploration to balance the trade-off between exploration and exploitation. Two different agents were created that chose actions (up, down, left, and right) randomly. One agent was a random agent that gave a baseline performance and the second agent was a Q-learning agent. We explored the effect of different values of epsilon (ε) and gamma (γ) on the Q-learning agent to analyze effects of exploration, exploitation, and future rewards.

We found that the random agent achieved a reward per episode that varied greatly from a range of -1000 to 0. However, the Q-learning agent converged to a high reward per episode at around 1000 episodes, specifically at a reward per episode of -4. We also found that the Q-learning agent's policy for all states varied drastically as different actions were used, allowing the dasher to reach the customer in a shorter distance. Thus, the Q-learning agent had significant improvements compared to the random agent.

When comparing Q-learning agents with different epsilon (ε) = 0.05 and 0.75, the agent with the larger epsilon (ε) value had more variability in the reward. This is because larger values of epsilon (ε) lead to more exploration and therefore more steps were wasted to get to the customer's location. We found that even though increasing the epsilon (ε) value decreased the performance of the Q-learning agent, increasing the epsilon (ε) value did not affect the performance of random agent. Additionally, we found that increasing the discount factor, or gamma (γ), decreases the delivery time of the Q-learning agent.

In the future, we could extend our work by implementing a deep Q-learning model to explore intricate and continuous state representations. Additionally, we could explore a Doordash environment that incorporates more dynamic variables like peak hours and traffic conditions.

Contributions

To implement the solution, all group members first worked independently and used Q-learning to solve the dasher problem over 2-3 weeks. Then, we came together and decided on a hybrid approach that incorporated ideas from each of our approaches. For the manuscript, Siya focused on producing and writing results, Karthik focused on developing a model using the methods mentioned, and Arman focused on the introduction/context, prior work, and abstract.

Acknowledgment

We would like to thank Mykel Kochenderfer and the AA228/CS238 Autumn 2023 teaching team for assisting us through the course and this project.

References

[1] Z. Li and G. Wang, "The Role of On-Demand Delivery Platforms in Restaurants," SSRN Electronic Journal, 2020, doi: https://doi.org/10.2139/ssrn.3813891.

[2] X. Chen, M. W. Ulmer, and B. W. Thomas, "Deep Q-learning for same-day delivery with vehicles and drones," European Journal of Operational Research, Jun. 2021, doi: https://doi.org/10.1016/j.ejor.2021.06.021.

[3] M. Silva, J. P. Pedroso, and A. Viana, "Deep reinforcement learning for stochastic last-mile delivery with crowdshipping," EURO Journal on Transportation and Logistics, vol. 12, p. 100105, 2023, doi: https://doi.org/10.1016/j.ejtl.2023.100105.

[4] A. Revadekar, R. Soni, and A. V. Nimkar, "QORAl: Q Learning based Delivery Optimization for Pharmacies," Jul. 2020, doi: https://doi.org/10.1109/icccnt49239.2020.9225589.

[5] C. Gao et al., "A Deep Learning Method for Route and Time Prediction in Food Delivery Service," Aug. 2021, doi: https://doi.org/10.1145/3447548.3467068.

[6] A. Adak, B. Pradhan, and N. Shukla, "Sentiment Analysis of Customer Reviews of Food Delivery Services Using Deep Learning and Explainable Artificial Intelligence: Systematic Review," Foods, vol. 11, no. 10, p. 1500, May 2022, doi: https://doi.org/10.3390/foods11101500.

[7] H. Park and K.-W. Oh, "A Path-Based Personalized Recommendation Using Q Learning," Transactions on computational science and computational intelligence, pp. 323–333, Jan. 2021, doi: https://doi.org/10.1007/978-3-030-70296-025.

[8] Z. Bi, X. Guo, J. Wang, S. Qin, and G. Liu, ”Deep Reinforcement Learning for Truck-Drone Delivery Problem,” Drones, vol. 7, no. 7, p. 445, Jul. 2023, doi: https://doi.org/10.3390/drones7070445.

[9] J. Wu and X. Xu, ”Decentralised grid scheduling approach based on multi-agent reinforcement learning and gossip mechanism,” CAAI Transactions on Intelligence Technology, vol. 3, no. 1, pp. 8–17, Mar. 2018, doi: https://doi.org/10.1049/trit.2018.0001.