Reinforcement Learning Urban Mobility

Report for the PhD admission procedure

Anastasia Psarou

Volos, Greece
2023-06-12

Contents

1 Introduction **2**

2 Problem Description **2**

3 User Equilibrium **3**

3.1 Centralized Solution 4

3.2 Decentralized Solution 6

3.3 Comparison 8

4 System Optimum **8**

4.1 Centralized Solution 9

4.2 Decentralized Solution 11

4.3 Comparison 12

5 Conclusion **13**

1 Introduction

In this report, a solution to an urban mobility problem is presented using reinforcement learning. Reinforcement learning is a machine learning approach that enables an agent to learn optimal decision-making by interacting with an environment and receiving feedback in the form of rewards or penalties.

In this project, the challenge of guiding travelers from specific origin points to their desired destinations is addressed. The available options for travel are two routes, namely path A and path B. Each path is associated with a distinct cost function, impacting the overall experience for travelers who choose that particular path. To ensure that travelers can identify the most favorable route, they must actively engage with the environment and make informed decisions. Consequently, the problem aligns with the principles of reinforcement learning, as the travelers navigate the environment and learn to optimize their decision-making process.

2 Problem Description

The problem about to be solved concerns the selection of paths between origins and destinations in transportation networks. In this particular scenario, the problem considers a total of Q travelers, with Q initially set to 1000 vehicles per hour. Additionally, there is the maximum number of vehicles allowed on path A, denoted as Q_a, with an initial value of 800 vehicles per hour, and the maximum number of vehicles on path B, denoted as Q_b, with an initial value of 500 vehicles per hour.

Furthermore, the concept of free flow speed is included, which represents the travel time when there is only one vehicle on the path. Specifically, the free flow speed for route A is denoted as t_a^0 and has a value of 5 minutes, while for route B it is denoted as t_b^0 and has a value of 15 minutes.

Finally, the q_a is defined as the number of vehicles currently traveling on route A, and q_b as the number of vehicles currently traveling on route B. Lastly, there is a cost function, which is defined using a naive, non-linearly increasing formula

$$t_a(q_a) = t_a^0 \left(1 + \left(\frac{q_a}{Q_a}\right)^2\right), \tag{1}$$

where $q_a + q_b = 1000$ and $q_a, q_b > 0$.

The goal of this problem is to create reinforcement learning environments to implement this environment using system optimum and user equilibrium as reward functions. User equilibrium and system optimum are important concepts in urban mobility because they contribute to the understanding of the behavior of traffic flow and its impact on transportation systems. Both concepts provide valuable insights into the efficiency, fairness, and overall performance of transportation networks.

User equilibrium is significant because it reflects the individual decision-making process of travelers. In a user equilibrium, each traveler selfishly chooses the path that appears most convenient to them, based on factors like travel time, cost, and personal preferences. This behavior accurately represents the real-world scenario where individuals make independent route choices. Achieving user equilibrium ensures fairness among users by guaranteeing that all travelers with the same origin and destination experience the same travel time. This fairness is important in maintaining public satisfaction and reducing potential conflicts or dissatisfaction among users.

On the other hand, the system optimum is crucial because it focuses on the overall efficiency of the transportation network. It seeks to minimize the total travel time for all users by optimizing the assignment of routes and traffic flows throughout the system. In the system optimum, traffic is routed in a way that reduces congestion, minimizes delays, and improves overall network performance.

3 User Equilibrium

User equilibrium is a fundamental principle, which states that for every origin and destination pair, all routes chosen by travelers should have equal and minimal travel time. In user equilibrium, each traveler seeks to minimize their own travel time, resulting in a state where no traveler can unilaterally improve their travel time by switching routes. This equilibrium represents a system where each individual traveler is satisfied with their chosen route based on their own preferences and constraints. This system is described by Equation 2

$$t_a(q_a) = t_b(q_b). \tag{2}$$

Based on Figure 1, it is evident that the point at which both routes have the same minimum travel time falls between 0.2 and 0.3. By performing further conservation of flow, we can determine that this point corresponds to 0.27. At this optimal value, the number of vehicles choosing path A, denoted as q_a, is equal to 755 and the number of vehicles choosing path B, denoted as q_a is equal to 245. Therefore, this equilibrium point represents the solution of our equation.

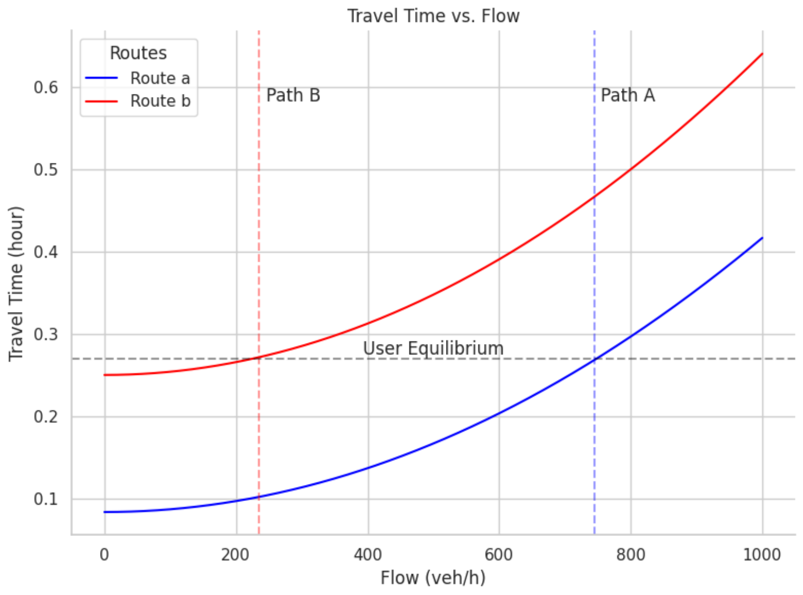

Figure 1: User equilibrium travel time with flow plot.

3.1 Centralized Solution

In the centralized solution, the problem is approached with two classes: *UserEquilibriumCentralizedEnvironment* representing the environment, and *UserEquilibrium-CentralizedController* representing the agent. The central controller assumes the role of training, and making decisions for every agent involved.

The central controller systematically evaluates all possible combinations of the number of vehicles choosing path A (denoted as q_a) and the number of vehicles choosing path B (denoted as q_a). It takes into account the limitations imposed on the values of q_a and q_a. For each combination, the controller calculates the associated reward by computing the absolute difference between the travel times on path A and path B, i.e., $|t_a(q_a) - t_b(q_b)|$. The goal is to find the combination of q_a and q_a that yields the minimum reward.

During each episode, the algorithm iterates over different values of q_a to determine the best action for the current state. The "choose_action" function is responsible for selecting an action either randomly or based on the current policy stored in the policy matrix.

After the action is determined, the reward is calculated based on the user equilibrium equation (Equation 2). Then, the next state is determined. In this problem, the state is represented by the values of q_a and q_b. Additionally, the policy array is updated based on these values. The policy dictates the actions that the agent takes as a function of its state and the environment.

If the current episode has the minimum reward, the value of the reward is saved along with the corresponding q_a value. The goal of this problem is to minimize the reward function. Furthermore, the values of q_a and q_b for each iteration are stored in lists to keep track of their changes throughout the training process.

```
def train(self, num_episodes):
    qa_values = []
    qb_values = []
    rewards = []
    best_reward = float('inf')
    best_qa = 0
    best_qb = 0

    for episode in range(num_episodes):
        self.env.reset()
        state = (self.env.qa, self.env.qb)
        min_reward = float('inf')

        for _ in range(self.env.Qa + 1):
            action = self.choose_action(state)
            reward = self.env.step(action)

            next_state = (self.env.qa, self.env.qb)

            self.update_policy(state, action)
            state = next_state

            if reward < min_reward:
                min_reward = reward
                min_qa, min_qb = self.env.qa, self.env.qb

        qa_values.append(min_qa)
        qb_values.append(min_qb)
        rewards.append(min_reward)

        # Check if the current episode achieved the best reward
        if min_reward < best_reward:
            best_reward = min_reward
            best_qa, best_qb = min_qa, min_qb

    return best_qa, best_qb
```

In Figure 2, the convergence of the reward function is observed over time, as well as the dynamic changes in the values of q_a and q_b across episodes. Notably, after 1000 episodes, the centralized solution achieves the minimum reward.

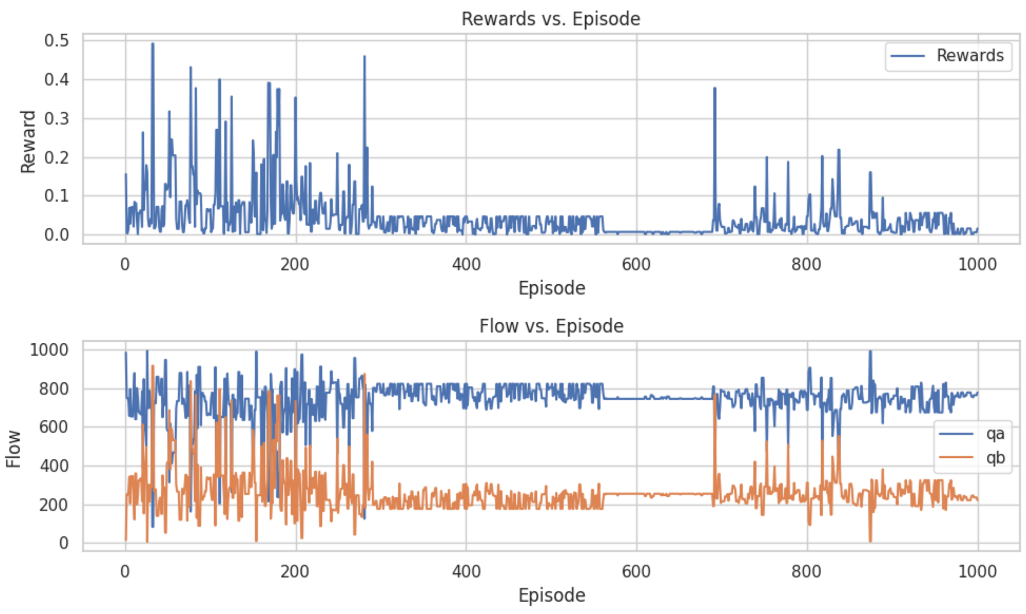

Figure 2: User equilibrium centralized approach results.

The graph clearly illustrates the progressive minimization of the reward function, indicating the effectiveness of the training process. Additionally, the varying values of q_a and q_b provide insights into the adaptive nature of the algorithm as it explores different actions in pursuit of optimal solutions.

It is worth mentioning that the time complexity of each episode can be specified as $O(Q_a)$, as the algorithm iterates over the different values of q_a during the decision-making process. This complexity allows for efficient training, even in scenarios with larger values of Q_a.

Overall, the results presented in Figure 2 highlight the successful convergence of the algorithm towards a user equilibrium solution and provide valuable information about the changes in system behavior throughout the training process.

3.2 Decentralized Solution

In the MARL (Multi-Agent Reinforcement Learning) solution, the *UserEquilibrum-MARLEnvironment* and *UserEquilibrumMARLAgents* classes are utilized. In the MARL solution, each agent selects an action randomly, but with the consideration of constraints to ensure that the values of q_a and q_b satisfy the conditions $Q_a < q_a$ and $Q_b < q_b$. If a randomly chosen action violates these constraints, it is adjusted or modified to maintain the validity of the constraints. This ensures that the chosen actions align with the capacity limitations imposed on paths A and B.

Once all agents have made their selections and the values of q_a and q_b have been determined, the corresponding reward is evaluated. If the reward for the current episode is lower than the minimum reward observed thus far, it becomes the new optimal episode. This process is repeated for a specified number of episodes.

After conducting 1000 episodes of training in the MARL solution, the system achieves a minimum reward of 0.143. This minimum reward value represents the best performance observed throughout the training process. Furthermore, the corresponding value of q_a associated with this minimum reward provides valuable information about the optimal solution for the system. By analyzing the minimum reward and its corresponding q_a value, insights can be gained regarding the most effective allocation of vehicles between paths A and B.

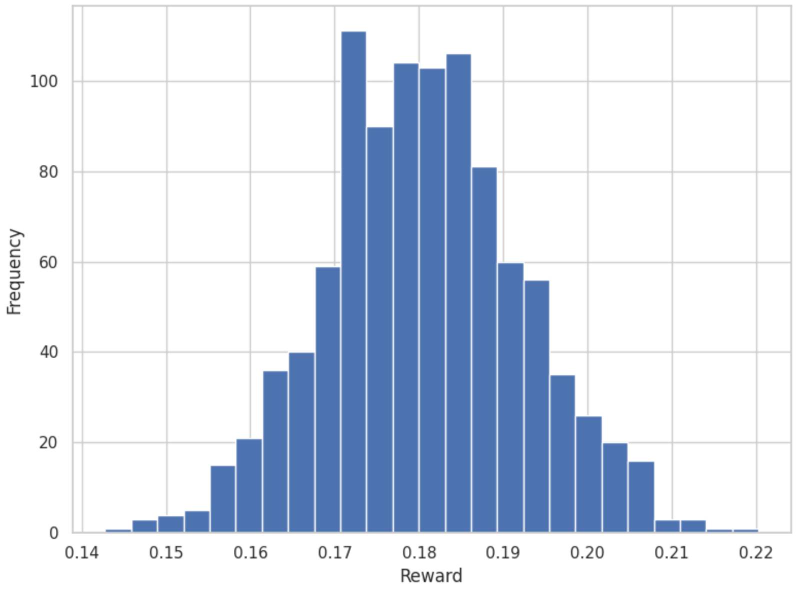

Figure 3: Histogram of user equilibrium decentralized approach.

The decentralized solution exhibits a complexity of $O(Q)$ for a single episode, where Q represents the total number of vehicles on both paths $(q_a + q_b)$. For multiple episodes, the overall complexity becomes $O(episodes * Q)$. However, it is important to note that the decentralized solution may not converge as closely to the optimal result compared to the centralized solution.

The first reason for that is that there was no training implemented in this solution between the environment and every agent individually. One more reason could be that agents make random choices between two classes of paths. As a result, it is more likely to have a distribution of vehicles where q_a and q_b are close to their initial values (in this case, 500). However, the optimal solution for the specific problem involves the value $q_a = 755$, which is not so easily reachable using a purely random algorithm.

For this reason, the histogram shown in Figure 3 exhibits a Gaussian-like distribution due to the random nature of the values. The Gaussian distribution, also known as the normal distribution, is a commonly observed probability distribution characterized by its symmetric bell-shaped curve. The presence of this distribution suggests that the values of q_a, q_b, and rewards tend to cluster around a central value, with fewer occurrences of extreme values.

3.3 Comparison

The comparison between the centralized and decentralized approaches in reaching the optimal solution for the system reveals the superiority of the centralized approach, as demonstrated by the results presented in Table 1. It is evident that the centralized solution outperforms the decentralized approach in terms of effectiveness and efficiency. Notably, the centralized solution exhibits faster convergence and achieves the optimal configuration in a significantly lower number of episodes. This remarkable performance advantage can be attributed to the absence of training in the MARL (Multi-Agent Reinforcement Learning) solution. Unlike the decentralized approach, the centralized solution benefits from a pre-defined strategy that optimizes the system's performance from the outset.

	q_a	q_b	**Minimum Reward**
Best Case	755	245	0.0000000001
Centralised	755	245	0.0001
Decentralised	553	447	0.142

Table 1: Comparison of the results of user equilibrium for 1000 episodes.

4 System Optimum

The concept of system optimum refers to an assignment strategy that minimizes the total system travel time. In our case, this optimization objective can be expressed using the equation:

$$t_a(q_a) \cdot q_a + t_b(q_b) \cdot q_b \tag{3}$$

where $t_a(q_a)$ and $t_b(q_b)$ represent the travel times on routes A and B, respectively. The goal is to minimize this expression subject to the constraints that the total number of vehicles is fixed, i.e., $q_a + q_b = Q$, and that the number of vehicles on each route cannot be negative, i.e., $q_a, q_b \geq 0$. By finding the values of q_a and q_b that minimize the above equation while satisfying the given constraints, we can achieve the system optimum, resulting in an efficient allocation of vehicles that minimizes the overall travel time.

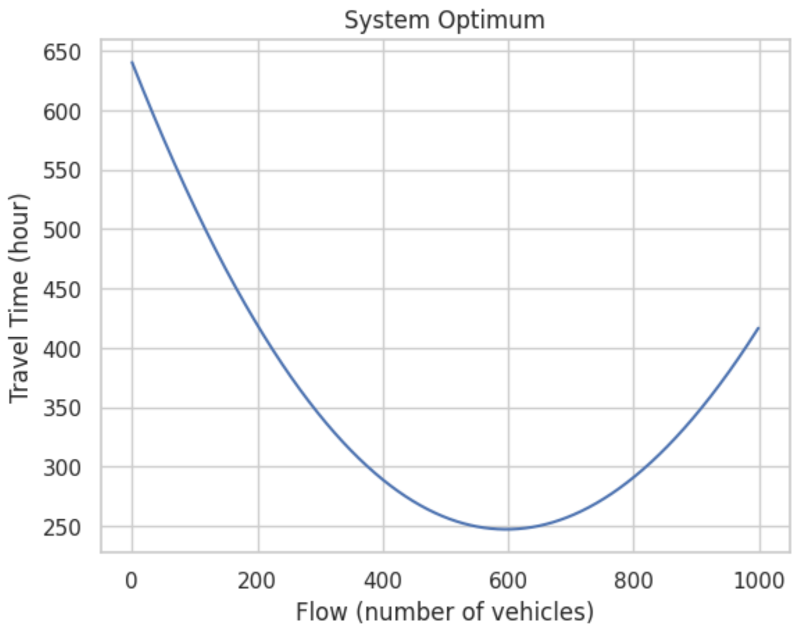

Figure 4: System optimum plot.

Based on the graph 4, the minimum travel time achievable for 1000 vehicles can be observed which is slightly less than 250 units, specifically measured at 246.99 hours. This value corresponds to the minimum of the equation 3, which represents the reward function of the problem. Additionally, for this minimum travel time, the optimal allocation of vehicles is found to be 597 vehicles on route a (q_a) and 403 vehicles on route b (q_b). These values are significant as they represent the desired outcome for our future solutions.

4.1 Centralized Solution

For the implementation of the centralized solution, the SystemOptimumCentralizedEnvironment and SystemOptimumCentralizedAgent classes were utilized. The SystemOptimumCentralizedEnvironment class represents the environment, while the SystemOptimumCentralizedAgent class is responsible for training the central controller, which assumes the role of collecting all relevant information, training, and making decisions for every agent involved.

Figure 5 illustrates the dynamic behavior of the reward over time, revealing a clear trend of convergence. As the training progresses, the reward gradually decreases, indicating an improvement in the system's performance. Additionally, the figure provides valuable insights into the number of q_a and q_b values utilized during the training process. By examining the plot, it becomes evident how the distribution of these values evolves, reflecting the agent's decision-making and resource allocation throughout the learning process.

The "choose_action" function is a critical component responsible for selecting an appropriate action based on a given state within a reinforcement learning algorithm. Its purpose is to strike a balance between exploration and exploitation strategies.

When a randomly generated number within the range of 0 to 1 is less than the exploration rate (epsilon), initially set to 0.005, the function opts for exploration. In this case, a random action is chosen, and the values of action_a and action_b, which represent the quantities of interest (q_a and q_b), are determined accordingly.

Conversely, when the generated number exceeds or equals epsilon, the function prioritizes exploitation by utilizing the current policy to select an action. The policy for the given state can be either a single integer or a tuple of integers representing the available actions. The policy_state value is assigned to action_a.

To ensure that action_a and action_b adhere to a minimum value of 1, the function applies a maximum operation on both variables. Additionally, it adjusts the values to guarantee their sum equals Q, effectively redistributing any remaining difference equally between the two actions.

Overall, the choose_action function plays a crucial role in the reinforcement learning process, intelligently navigating the exploration-exploitation trade-off to facilitate effective decision-making based on the provided state information.

```
def choose_action(self, state):
        if np.random.uniform(0, 1) < self.epsilon:
            # Explore - choose a random action
            action_a = np.random.randint(1, self.env.Q)
            action_b = self.env.Q - action_a
        else:
            # Exploit - choose the action based on the current policy
            policy_state = self.policy[state[0], state[1]]
            if isinstance(policy_state, int):
                action_a = policy_state
            else:
                action_a = policy_state[0]
            action_b = self.env.Q - action_a

            action_a = max(1, action_a)
            action_b = max(1, action_b)
            diff = self.env.Q - (action_a + action_b)
            action_a += diff // 2
            action_b += diff // 2

        return action_a, action_b
```

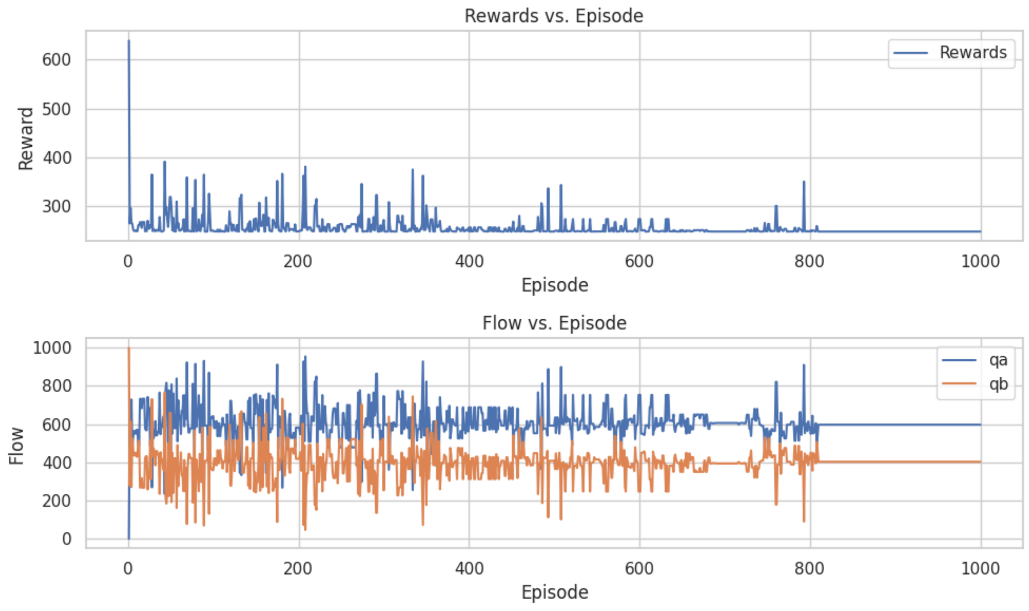

Figure 5: System optimum centralized results.

It is important to note that the time complexity of each episode in this algorithm, also, can be described as $O(Q_a)$, where Q_a represents the range of possible values for q_a. This complexity arises from the iteration over the different values of q_a during the decision-making process. This efficient time complexity enables effective training, even in scenarios with larger values of Q_a, ensuring the scalability and feasibility of the algorithm in practical applications.

4.2 Decentralized Solution

In the MARL (Multi-Agent Reinforcement Learning) solution, the SystemOptimum-MARLEnvironment and SystemOptimumMARLAgents classes are utilized.

During each episode, 1000 agents autonomously select their routes between a and b, while adhering to the capacity limitations denoted by Q_a and Q_b. Initially, the agents make random decisions, but the environment intervenes if the imposed constraints, $Q_a < q_a$ or $Q_b < q_b$, are violated, leading to adjustments in the agents' choices.

Upon completion of the agents' selections, the resulting values of q_a and q_a are computed, enabling the determination of the minimum attainable reward as prescribed by the objective function 3.

Figure 6 provides a histogram depicting the frequency distribution of the reward values. The histogram exhibits a gaussian distribution, indicating that the rewards

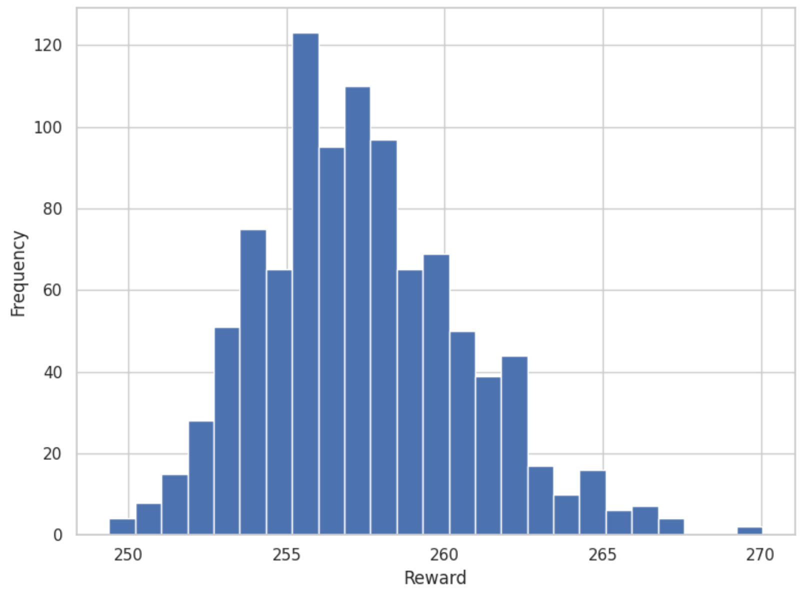

Figure 6: Histogram of the system optimum decentralized solution.

are clustered around a certain mean value.

In the decentralized system optimum solution, a closer convergence to the optimal solution is observed compared to the decentralized user equilibrium solution. This can be attributed to the fact that the minimal reward, corresponding to $q_a = 597$, is relatively easier to achieve through random exploration between two classes compared to the user equilibrium solution where the minimal reward corresponds to $q_a = 755$.

In summary, the decentralized solution exhibits a complexity of $O(Q)$ per episode, where Q represents the total number of vehicles on both paths (q_a+q_b). With multiple episodes, the overall complexity increases to $O(episodes*Q)$. However, it is important to note that the decentralized approach may not converge as closely to the optimal result compared to the centralized solution.

4.3 Comparison

Table 2 presents a comparison of the results obtained from centralized and decentralized approaches on the system optimum solution. Through the analysis of these results, clear patterns emerge, highlighting the superiority of the centralized solution over the decentralized counterparts in terms of efficiency and speed in achieving the final optimal configuration.

The centralized approach consistently outperforms the decentralized solutions in terms of convergence and efficiency. It demonstrates a remarkable ability to quickly converge towards the optimal solution, regardless of the number of episodes. On the other hand, the decentralized solutions, with varying numbers of episodes, exhibit

	q_a	q_b	**Minimum Reward**
Best Case	597	433	246.99
Centralised	597	433	246.99
Decentralised	550	450	249.38

Table 2: Comparison of the results of the system optimum environment for 1000 episodes.

a slightly slower convergence rate. It is important to note that the decentralized solutions were produced by random choices of each agent without undergoing any training from the environment.

It should be emphasized that the lack of training in the decentralized solution is a significant factor contributing to its slower convergence. If training were implemented, the results could potentially differ, and the decentralized solution may exhibit improved convergence and performance.

5 Conclusion

In conclusion, both centralized solutions, namely the central controller in the system optimum approach and the centralized MARL agents, demonstrate good performance and quite similar results between them. The choice between them depends on the specific goals of the problem creator. In terms of efficiency, both centralized solutions exhibit satisfactory performance, achieving close approximations to the optimal solution.

In contrast, the decentralized approaches, specifically the user equilibrium solution, exhibit less effective convergence compared to the system optimum solution. This discrepancy arises from the challenge of specifying precise values for q_a and q_b in the user equilibrium approach. The system optimum approach, on the other hand, has the advantage of being able to explore a broader range of values and identify solutions that closely approximate the optimal value within the given environment. Therefore, when aiming for precise convergence to specific values, the system optimum approach proves to be more reliable and advantageous.

References

[1] Centralized Training and Decentralized Execution in Multi-Agent Reinforcement Learning

[2] Decentralized Reinforcement Learning

[3] User equilibrium and system optimum