Reinforcement Learning in Urban Mobility

Ph.D. Qualification Project

Ahmet Onur Akman

September 2023

Contents

Chapter 1

Introduction

Optimization of urban mobility is an increasingly important challenge in today's world. As cities expand and the number of travelers in the city traffic grows, optimizing route choices becomes a crucial task. The essence of this problem lies in balancing the system-level optimal solutions with user-level optimal solutions. This is a complex task, because of the dynamic nature of travel behavior and the intricacies involved in route selection.

The goal of this study is to explore and analyze how travelers make choices between alternative routes from an origin O to a destination D and how these choices impact the overall system performance. The problem sets up a formal problem with two main alternative routes a and b, governed by a non-linearly increasing BPR (Bureau of Public Roads) travel time function.

The problem is initially tackled through classical optimization methods to compute both the System Optimum (SO) and User Equilibrium (UE), providing a foundational understanding of the system dynamics. This sets the stage for the application of Reinforcement Learning (RL) methods to model the decision-making process of individual travelers.

The study adopts a multi-step approach:

- Analytical and numerical computation of SO and UE.
- Formulation and solution of the problem using basic RL methods.
- Extension of the problem to include additional complexities, such as stochastic events and extra routes. Application of deep RL algorithms to optimize the traffic flow under new constraints.
- Further extension of the problem, and a proposed solution using RL algorithms.

The report contains visualizations and computational models, implemented in Python notebooks, to aid the understanding and interpretation of the results. Each phase of the problem and its solution is documented to provide a comprehensive view of the subject.

1.1 Problem

We explore a transportation system containing two routes (a and b) connecting an origin (O) to a destination (D). A set of Q individual travelers must make a decision on which route to take, based on the travel time and their preferences.

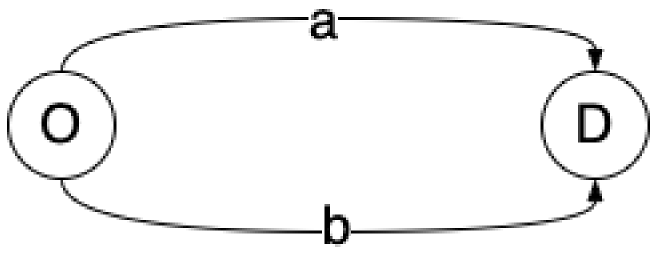


Figure 1.1: System definition

The travel time on each arc (a or b) is represented by a non-linearly increasing BPR formula:
$t_a(q_a) = t_a^0 \left(1 + \left(\frac{q_a}{Q_a}\right)^2\right)$
where:

- $t_a(q_a)$ - is the travel time on arc a (or b)
- q_a - is the flow (number of vehicles using arc)
- t_a^0 - is the free flow speed (with no other vehicles)
- Q_a - is the capacity (maximal number of vehicles)

Following the parameterization:

- Q = 1000 veh/h
- t_a^0 = 5 min
- t_b^0 = 15 min
- Q_a = 500 veh/h
- Q_b = 800 veh/h

First off, the analytic computations of SO and UE are presented, where:

1. **System Optimum (SO)**: Optimal distribution of travelers between routes a and b that minimizes the total cost (travel time) across all individuals. We seek to minimize:

$t_a(q_a) \cdot q_a + t_b(q_b) \cdot q_b$
s.t. $q_a + q_b = Q$ and $q_a, q_b \geq 0$

2. **User Equilibrium (UE)**: The configuration where every traveler is individually satisfied with their choice, meaning that the travel time on both routes is equal:

$t_a(q_a) = t_b(q_b)$

Later, the problem of finding SO and UE is modeled in a RL system, and these configurations are once again obtained in a centralized and multi-agent setting.

Upon this, the system was modified, so that it became ideal for a deep reinforcement learning-based solution. We explored the limits and the gains of this new approach.

Finally, we present a new problem, the traffic optimization challenge for a city network. We call this imaginary city Driveington, and the locals of this city try to get to their destinations from their initial points, at the same time, and as fast as possible. We propose another RL-based solution for such a system and try to see if our approach is still satisfactory even when the problem complexity is increased.

Chapter 2

Analytical and Numerical Calculations of the SO and the UE Configurations

2.1 Exploratory Analysis

First, we try to have an understanding of the problem with some exploration in our system. Figure-2.1 shows how the travel time changes with the number of vehicles for each route. Figure-2.2 shows how the system total cost changes with the assigned number of vehicles to Route A.

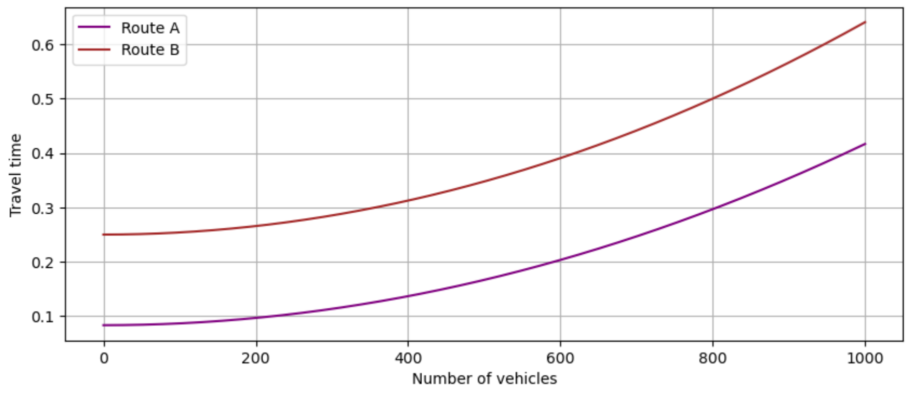


Figure 2.1: For each route, how the travel time changes with the number of vehicles using it.

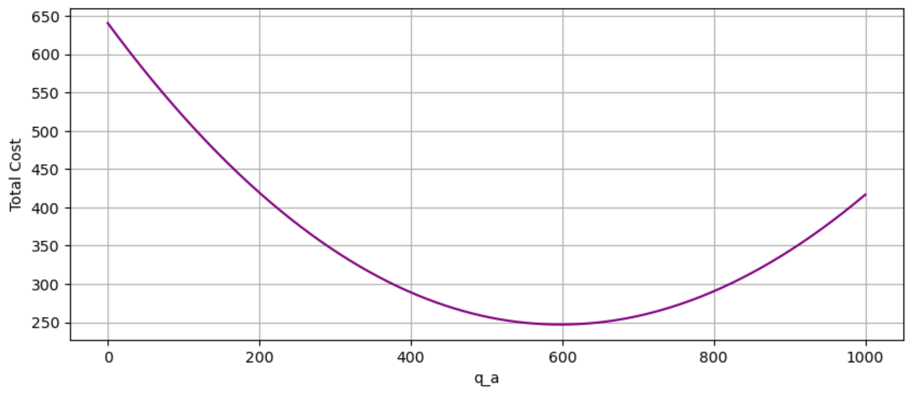


Figure 2.2: Following the constraints, how the system total cost changes with the assigned number of vehicles to Route A.

From Figure-2.2, we understand that the SO state is where $q_a = 597$ and $q_b = 403$. We will see if we can reach this solution with our methods.

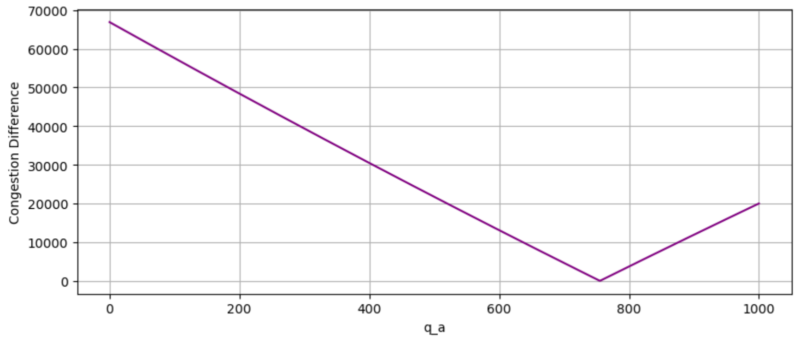


Figure 2.3: Following the constraints, how the cost difference of two routes changes with the assigned number of vehicles to Route A.

From Figure-2.3, we understand that the UE state is where $q_a = 755$ and $q_b = 245$. We will see if we can reach this solution with our methods.

2.2 Analytical Solution for the System Optimality

Following is the analytical solution for the SO configuration.

$TotalCost = t_a(q_a) \cdot q_a + t_b(q_b) \cdot q_b$
Given $q_b = Q - q_a$,
$TotalCost = t_a(q_a) \cdot q_a + t_b(Q - q_a) \cdot (Q - q_a)$
Given parameterization,

$TotalCost = \frac{5}{60} \cdot \left(1 + \left(\frac{q_a}{500}\right)^2\right) \cdot q_a + \frac{15}{60} \cdot \left(1 + \left(\frac{(1000-q_a)}{800}\right)^2\right) \cdot (1000 - q_a)$

Simplifying Total Cost

$\frac{1}{12} \cdot (1 + \frac{q_a^2}{25 \cdot 10^4}) \cdot q_a + (\frac{1}{4} \cdot 1 + \frac{(10^3-q_a)^2}{64 \cdot 10^4}) \cdot (10^3 - q_a) =$

$(\frac{q_a}{12} + \frac{q_a^3}{3 \cdot 10^6}) + (\frac{10^3-q_a}{4} \cdot (1 + \frac{(10^3-q_a)^2}{64 \cdot 10^4})) =$

$\frac{q_a}{12} - \frac{q_a}{4} + \frac{q_a^3}{3 \cdot 10^6} + \frac{(10^3-q_a)^3}{256 \cdot 10^4} + \frac{10^3}{4}$

Derivative

$\frac{1}{12} - \frac{1}{4} + \frac{q_a^2}{10^6} - \frac{3 \cdot (10^3-q_a)^2}{256 \cdot 10^4} =$

$\frac{-1}{6} + \frac{q_a^2}{10^6} + \frac{(-3 \cdot 10^6)+(6 \cdot 10^3 \cdot q_a)-(3 \cdot q_a)}{256 \cdot 10^4} =$

$\frac{-1}{6} + \frac{q_a^2}{10^6} - \frac{300}{256} + \frac{6 \cdot q_a}{2560} - \frac{3 \cdot q_a^2}{256 \cdot 10^4}$

Simplifies to:

$-\frac{33 \cdot q_a^2 - 450000 \cdot q_a + 257000000}{192000000}$

Total Cost is minimized where $\frac{\partial \textit{Total Cost}}{\partial q_a} = 0$

$-33q_a^2 + 45 \cdot 10^4 \cdot q_a - 257 \cdot 10^6 = 0$

$\Delta = (45 \cdot 10^4)^2 - 4 \cdot (-33) \cdot (-257 \cdot 10^6)$

$\Delta = 168576000000$

Delta is greater than zero.

$\sqrt{\Delta} = 410580.077$

$q_{a1} = \frac{-(45 \cdot 10^4) - \sqrt{\Delta}}{2 \cdot (-33)}$ and $q_{a2} = \frac{-(45 \cdot 10^4) + \sqrt{\Delta}}{2 \cdot (-33)}$

$q_{a1} \approx 13039.09$ and $q_{a2} \approx 597.27$

Given $q_a \leq Q$, $q_a \geq 0$ and $q_a \epsilon \mathbb{N}$

$q_a \approx 597$ and $q_b \approx 403$

2.3 Numerical Solution for the System Optimality

Leveraging from first-order and second-order derivatives of the cost function, we also computed the solution using the Newton-Raphson method.

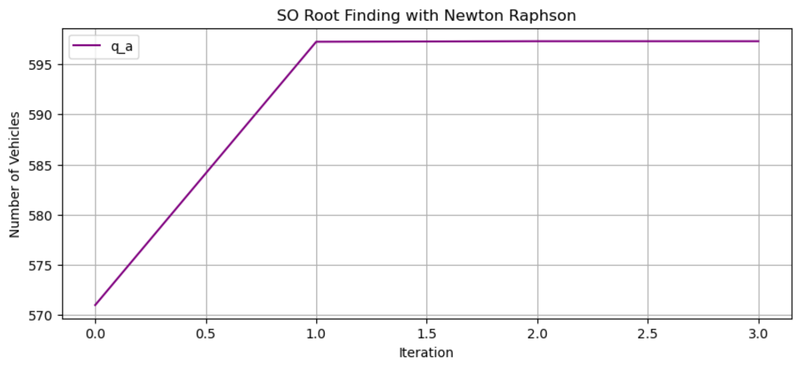


Figure 2.4: SO Root Finding with Newton Raphson

The algorithm successfully finds the solution $q_a \approx 597$ and $q_b \approx 403$ in 3 iterations.

2.4 Analytical Solution for the User Equilibrium

Following is the analytical solution for the UE configuration.

$t_a(q_a) = t_b(q_b)$
$t_a^0(1 + (\frac{q_a}{Q_a})^2) = t_b^0(1 + (\frac{q_b}{Q_b})^2)$
$t_a^0(1 + (\frac{q_a}{Q_a})^2) = t_b^0(1 + (\frac{Q-q_a}{Q_b})^2)$
$\frac{5}{60}(1 + (\frac{q_a}{500})^2) = \frac{15}{60}(1 + (\frac{1000-q_a}{800})^2)$
$\frac{q_a^2}{25 \cdot 10^4} = 2 + \frac{3 \cdot (1000-q_a)^2}{64 \cdot 10^4}$
$\frac{q_a^2}{25} = \frac{(3 \cdot q_a^2)-(6 \cdot 10^3 \cdot q_a)+(128 \cdot 10^4+3 \cdot 10^6)}{64}$
$(\frac{11}{1600}) \cdot q_a^2 + (\frac{-375}{4}) \cdot q_a + 66875 = 0$
$\Delta = (\frac{-375}{4})^2 - 4 \cdot \frac{11}{1600} \cdot 66875 = 6950$
$\sqrt{\Delta} \approx 83.36666$
$q_{a1} = \frac{-\frac{375}{4}-\sqrt{\Delta}}{2 \cdot \frac{11}{1600}} \approx 755.15$
$q_{a2} = \frac{-\frac{375}{4}+\sqrt{\Delta}}{2 \cdot \frac{11}{1600}} \approx 12881.21$
Given $q_a \leq Q$, $q_a \geq 0$ and $q_a \epsilon \mathbb{N}$
$q_a \approx 755$ and $q_b \approx 245$

2.5 Numerical Solution for the User Equilibrium

We also computed the solution using the Successive Averages Method.

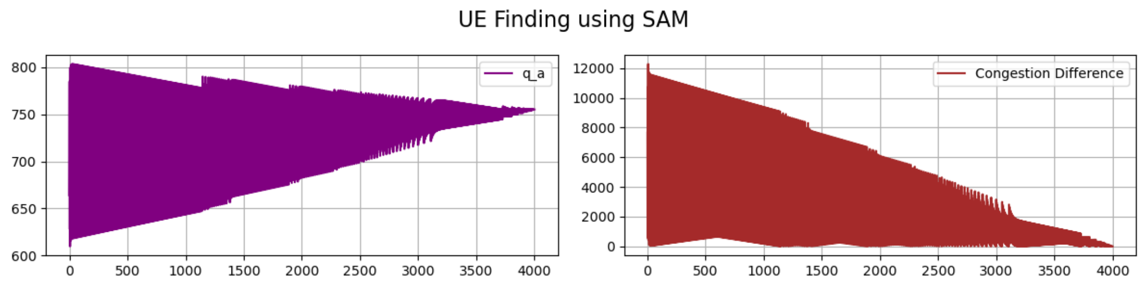


Figure 2.5: UE Finding with SAM

SAM successfully finds the configuration $q_a \approx 755$ and $q_b \approx 245$ in about 4000 iterations.

Chapter 3

Reformulation of the Problem as an RL System: Centralized Approach

3.1 Introduction

In this chapter, we aim to extend our analytical insights into optimizing traffic flow by employing Reinforcement Learning (RL) techniques. We will focus on:

- Formulating our traffic optimization problem as an RL problem, using states, actions, policies, and rewards.
- Implementing our formulation using RL framework.
- Comparing the RL-based solutions with the analytical results to validate the efficacy of our RL approach.

Our ultimate objective is to demonstrate how RL can be effectively used to solve complex, real-world problems like traffic flow optimization.

At this step, since we are focused on finding a model that can effectively converge to the system optimum and user equilibirum, we will have a centralized approach and we will assume that **the entire traffic flow is managed by one central agent**. However, we do not only aim for our central agent to find a policy that leads to an eventual satisfactory result under the parameterization, **we aim to create a traffic control agent which has an understanding about what kind of decision making is needed for balancing the traffic flow at every step of the episode**.

3.2 Remodeling the System as a RL Problem

3.2.1 Environment

We have designed an environment that is simple yet able to capture all the requirements imposed by the problem definition. The overall structure of the environment for both SO and UE are very similar, they differ in their reward functions. They are both designed to accept binary actions, for adding a single car on Route A or Route B. Initially, both roads are empty. The environments finalize the episode when $q_a + q_b = Q$.

3.2.2 State Space

The discrete state space is made of tuples (q_a, q_b) representing the current number of vehicles on each route. It is a discrete state space as this aligns with the discrete nature of vehicle counts. Transitions between states are defined only in terms of the addition of a single car on one of the routes, therefore to keep everything more realistic, it is made impossible to decrease the number of cars in a route once it is added. Keeping the state space discrete makes the problem tractable and well-suited for a table look-up policy.

3.2.3 Action Space

The action space consists of only two actions:

- Add one vehicle to route ‘A‘ (increase q_a by one)
- Add one vehicle to route ‘B‘ (increase q_b by one)

3.2.4 Agent

The agent uses a deterministic table look-up policy and employs a Q-Learning algorithm to learn the best actions to take in various states. Therefore we are more interested in the value of taking an action a in a state s, rather than assigning values for each state.

Given the discrete and finite nature of the state and action spaces, a table look-up policy offers an efficient way to store and retrieve the optimal policy. Q-learning allows the agent to learn optimal policies efficiently.

The deterministic lookup policy might create problems in exploration/exploitation, therefore there is an epsilon term to occasionally make random decisions so that a good portion of the state space is explored. The learning process involves epsilon scheduling so that after we do the necessary exploration, we can exploit the best states and tune our quality function further.

3.2.5 Reward Function

The reward function is the part that enables an agent to learn how to handle traffic flow efficiently in **every step in an episode**, **rather than reaching to a SO or UE after a set of random actions**.

For instance, if we know that we have system optimality when $q_a = x$ and $q_b = y$, the agent should not keep adding cars to one of the routes and then proceed with the other one. This will create high congestion in one of the routes while the other route is completely empty. Rather, **the agent should be mindful and add cars in such a way that in every step of the episode the congestions in both of the routes are ideal**.

Also, in order to utilize every training episode and handle the credit assignment problem, one needs to make sure the rewards are *dense* enough.

To achieve this, I built reward functions that are essentially the given formulas but with few tricks.

For the **SO** task, the reward function is defined as follows:

$curr_Q = q_a + q_b$
$k1 = q_a + ((Q - curr_Q) * \frac{q_a}{curr_Q})$
$k2 = q_b + ((Q - curr_Q) * \frac{q_b}{curr_Q})$
$reward = -(t_a(k1) \cdot k1 + t_b(k2) \cdot k2) * \frac{curr_Q}{Q}$

For the **UE** task, the reward function is defined as follows:

$curr_Q = q_a + q_b$
$k1 = q_a + ((Q - curr_Q) * \frac{q_a}{curr_Q})$
$k2 = q_b + ((Q - curr_Q) * \frac{q_b}{curr_Q})$
$reward = -abs(t_a(k1) - t_b(k2)) * \frac{curr_Q}{Q}$

The reward function is carefully designed to keep the system close to optimal given the goal in every step, but much more carefully as we approach the end of the episode. This method ensures that the agent looks ahead and makes decisions that are beneficial in the long run. It can be seen that **the quality of the final state is much more pronounced**, this is for assuring the convergence to the solution of our problem.

3.3 Computational Complexity

3.3.1 Environment Complexity

- step method: $O(1)$
- calculate_reward: $O(1)$

3.3.2 Agent Complexity

- choose_action: $O(1)$ (Remember: Only 2 possible actions)
- update_q_value: $O(1)$ (Remember: Only 2 possible actions)
- decay_epsilon: $O(1)$

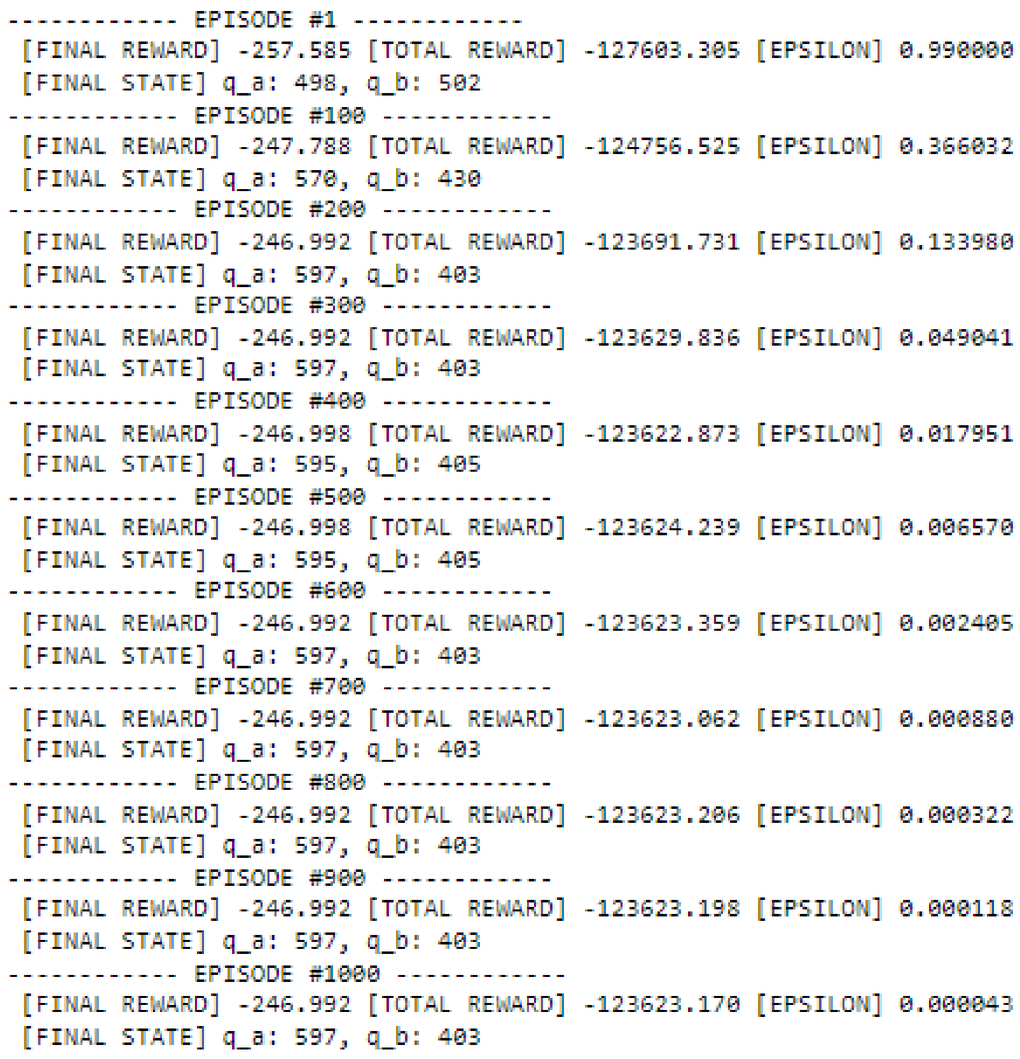
3.3.3 Training Loop Complexity

- Outer loop (num_episodes): $O(N)$, where N is the number of episodes.
- Inner loop ('while not done'): Worst case, this will run for $O(Q)$ steps, where Q is the number of vehicles.

The overall complexity is $O(N \times Q)$.

3.4 SO: Training and Stats

The training is configured with 1000 episodes, with $epsilon = 1.0$. Epsilon is decayed with a multiplier of 0.99 at the end of each episode.



```
------------ EPISODE #1 ------------
 [FINAL REWARD] -257.585 [TOTAL REWARD] -127603.305 [EPSILON] 0.990000
 [FINAL STATE] q_a: 498, q_b: 502
------------ EPISODE #100 ------------
 [FINAL REWARD] -247.788 [TOTAL REWARD] -124756.525 [EPSILON] 0.366032
 [FINAL STATE] q_a: 570, q_b: 430
------------ EPISODE #200 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -123691.731 [EPSILON] 0.133980
 [FINAL STATE] q_a: 597, q_b: 403
------------ EPISODE #300 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -123629.836 [EPSILON] 0.049041
 [FINAL STATE] q_a: 597, q_b: 403
------------ EPISODE #400 ------------
 [FINAL REWARD] -246.998 [TOTAL REWARD] -123622.873 [EPSILON] 0.017951
 [FINAL STATE] q_a: 595, q_b: 405
------------ EPISODE #500 ------------
 [FINAL REWARD] -246.998 [TOTAL REWARD] -123624.239 [EPSILON] 0.006570
 [FINAL STATE] q_a: 595, q_b: 405
------------ EPISODE #600 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -123623.359 [EPSILON] 0.002405
 [FINAL STATE] q_a: 597, q_b: 403
------------ EPISODE #700 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -123623.062 [EPSILON] 0.000880
 [FINAL STATE] q_a: 597, q_b: 403
------------ EPISODE #800 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -123623.206 [EPSILON] 0.000322
 [FINAL STATE] q_a: 597, q_b: 403
------------ EPISODE #900 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -123623.198 [EPSILON] 0.000118
 [FINAL STATE] q_a: 597, q_b: 403
------------ EPISODE #1000 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -123623.170 [EPSILON] 0.000043
 [FINAL STATE] q_a: 597, q_b: 403
```

Figure 3.1: Training logs

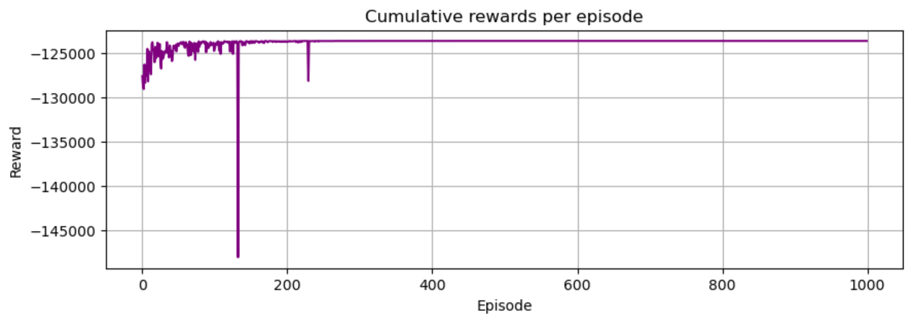


Figure 3.2: Episode rewards collected throughout the training

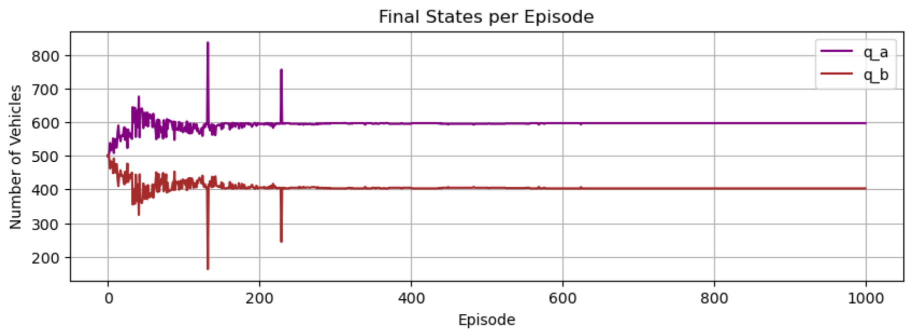
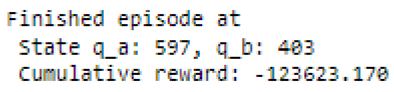


Figure 3.3: Final states at each episode

Figures-3.1, 3.2 and 3.3 show that the training successfully converges to the correct solution. The trend followed throughout the training also shows the efficiency of the training.

3.5 SO: Testing the Agent

We test the trained agent for one episode with fully deterministic decision-making.



```
Finished episode at
 State q_a: 597, q_b: 403
 Cumulative reward: -123623.170
```

Figure 3.4: Final state at the end of the episode

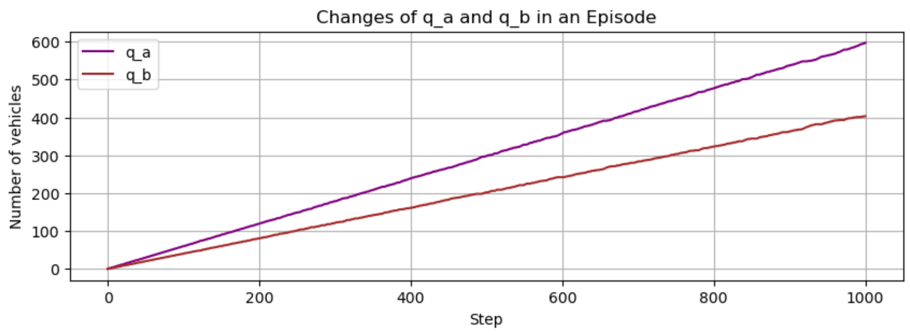


Figure 3.5: q_a and q_b throughout the episode.

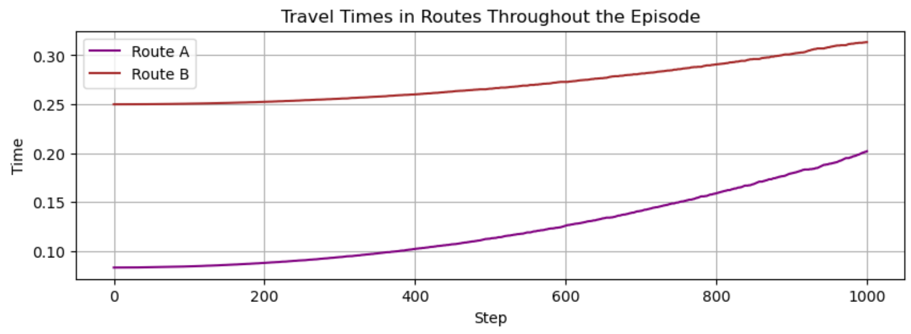


Figure 3.6: t_a and t_b throughout the episode.

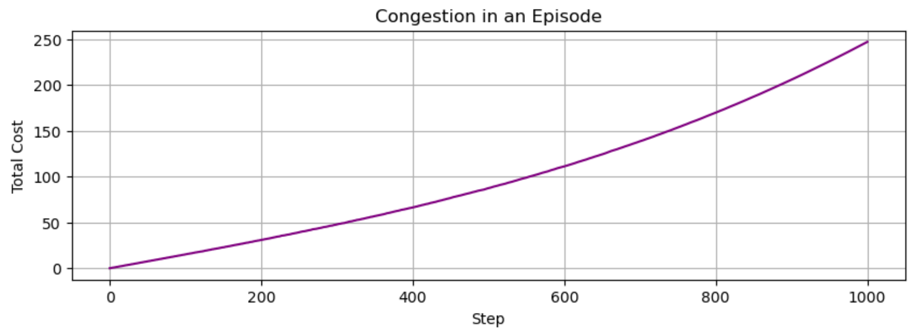
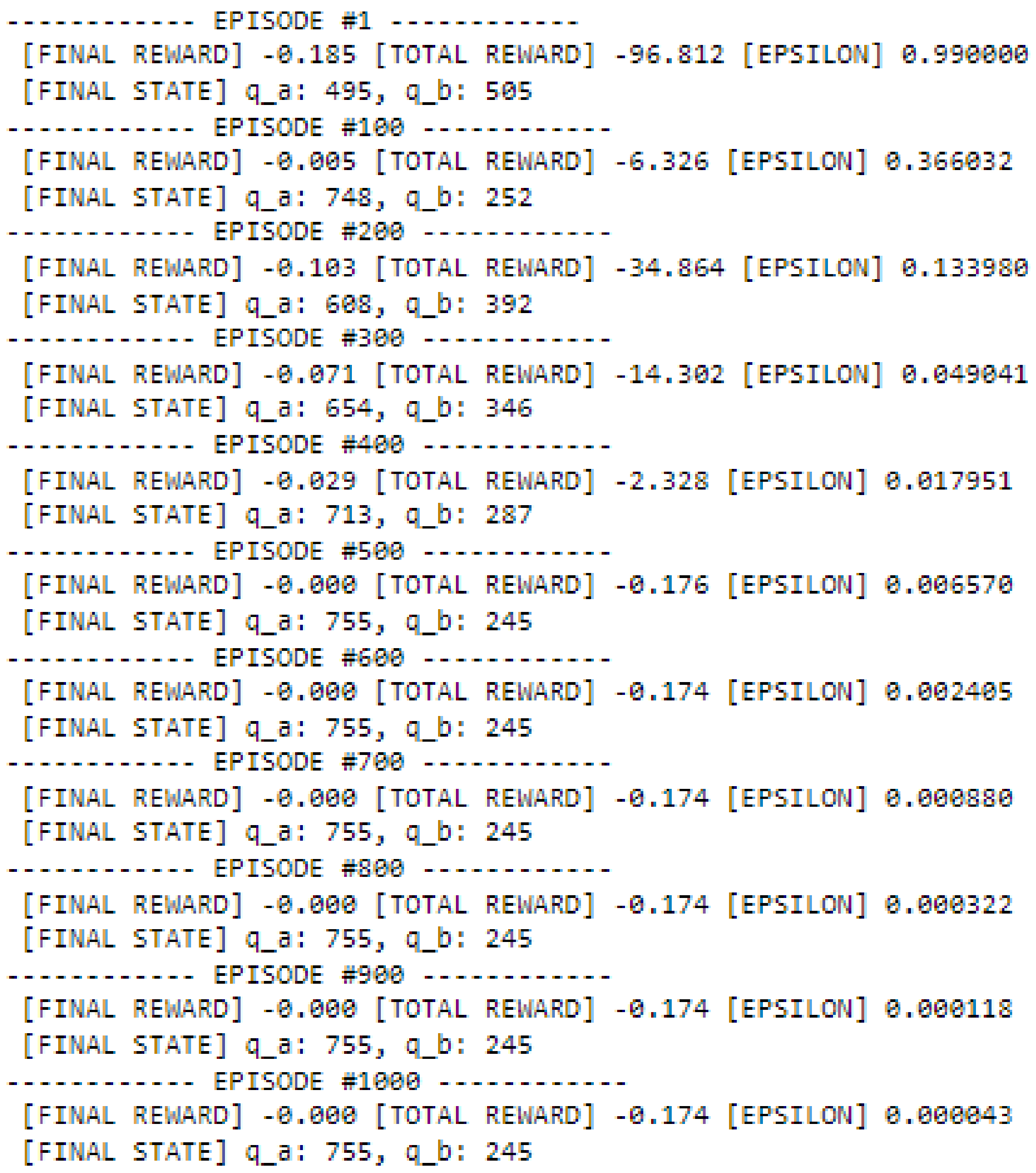


Figure 3.7: Total cost throughout the episode.

Our agent finds the SO solution at the end of the episode. Not only that, we also see that throughout the episode the agent decides carefully to keep the total cost low at all times.

3.6 UE: Training and Stats

The training is configured with 1000 episodes, with $epsilon = 1.0$. Epsilon is decayed with a multiplier of 0.99 at the end of each episode.



```
------------ EPISODE #1 ------------
 [FINAL REWARD] -0.185 [TOTAL REWARD] -96.812 [EPSILON] 0.990000
 [FINAL STATE] q_a: 495, q_b: 505
------------ EPISODE #100 ------------
 [FINAL REWARD] -0.005 [TOTAL REWARD] -6.326 [EPSILON] 0.366032
 [FINAL STATE] q_a: 748, q_b: 252
------------ EPISODE #200 ------------
 [FINAL REWARD] -0.103 [TOTAL REWARD] -34.864 [EPSILON] 0.133980
 [FINAL STATE] q_a: 608, q_b: 392
------------ EPISODE #300 ------------
 [FINAL REWARD] -0.071 [TOTAL REWARD] -14.302 [EPSILON] 0.049041
 [FINAL STATE] q_a: 654, q_b: 346
------------ EPISODE #400 ------------
 [FINAL REWARD] -0.029 [TOTAL REWARD] -2.328 [EPSILON] 0.017951
 [FINAL STATE] q_a: 713, q_b: 287
------------ EPISODE #500 ------------
 [FINAL REWARD] -0.000 [TOTAL REWARD] -0.176 [EPSILON] 0.006570
 [FINAL STATE] q_a: 755, q_b: 245
------------ EPISODE #600 ------------
 [FINAL REWARD] -0.000 [TOTAL REWARD] -0.174 [EPSILON] 0.002405
 [FINAL STATE] q_a: 755, q_b: 245
------------ EPISODE #700 ------------
 [FINAL REWARD] -0.000 [TOTAL REWARD] -0.174 [EPSILON] 0.000880
 [FINAL STATE] q_a: 755, q_b: 245
------------ EPISODE #800 ------------
 [FINAL REWARD] -0.000 [TOTAL REWARD] -0.174 [EPSILON] 0.000322
 [FINAL STATE] q_a: 755, q_b: 245
------------ EPISODE #900 ------------
 [FINAL REWARD] -0.000 [TOTAL REWARD] -0.174 [EPSILON] 0.000118
 [FINAL STATE] q_a: 755, q_b: 245
------------ EPISODE #1000 ------------
 [FINAL REWARD] -0.000 [TOTAL REWARD] -0.174 [EPSILON] 0.000043
 [FINAL STATE] q_a: 755, q_b: 245
```

Figure 3.8: Training logs

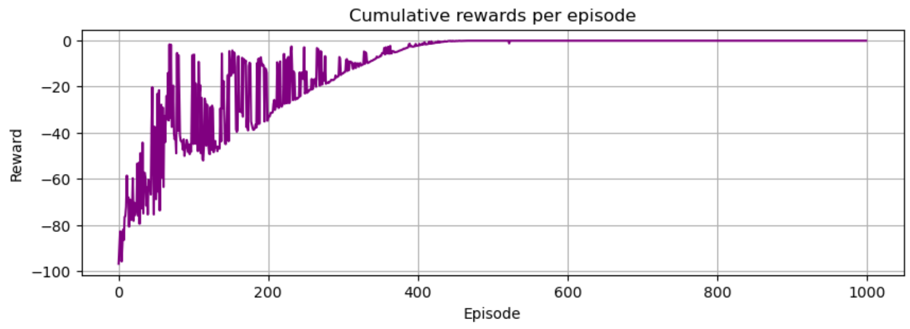


Figure 3.9: Episode rewards collected throughout the training

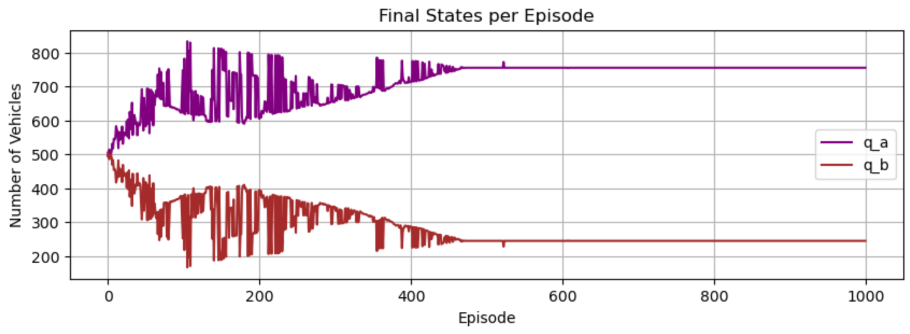
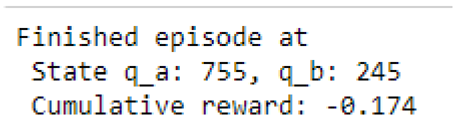


Figure 3.10: Final states at each episode

Figures-3.8, 3.9 and 3.10 show that the training successfully converges to the correct solution. The trend followed throughout the training also shows the efficiency of the training.

3.7 UE: Testing the Agent

We test the agent for one episode with fully deterministic decision-making.



```
Finished episode at
 State q_a: 755, q_b: 245
 Cumulative reward: -0.174
```

Figure 3.11: Final state at the end of the episode

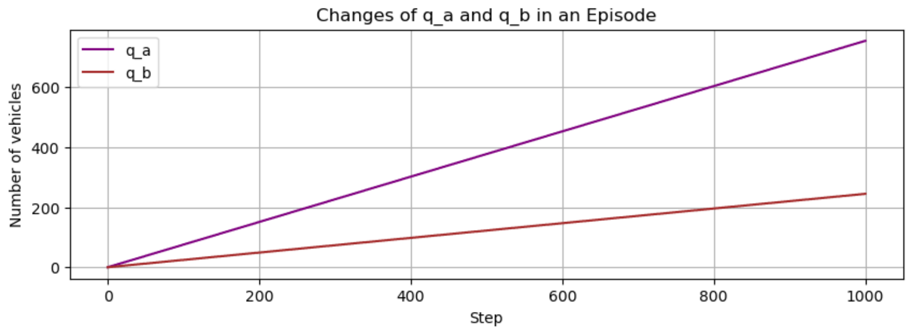


Figure 3.12: q_a and q_b throughout the episode.

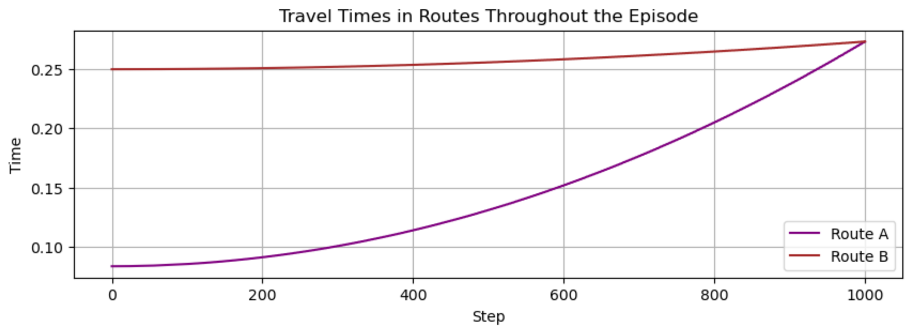


Figure 3.13: t_a and t_b throughout the episode.

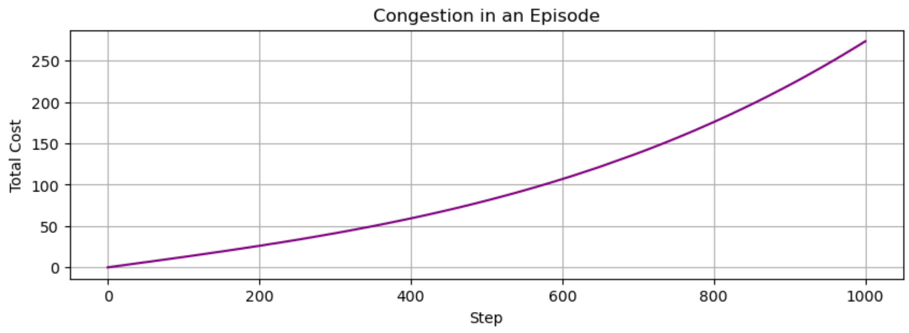


Figure 3.14: Total cost throughout the episode.

Our agent finds the UE solution at the end of the episode. Not only that, we also see that throughout the episode the agent decides carefully to keep the travel times on both routes similar at all times.

3.8 Conclusion

We have successfully modeled this traffic optimization problem using Reinforcement Learning. **Our traffic controller agents are capable of not just finding the System Optimum (SO) and User Equilibrium (UE) but also do so while considering incremental car distribution throughout every episode.**

- **Discrete State Space**: Made the problem more manageable and ideal for a look-up table policy.
- **Reward Function**: Designed to consider both the final goal and the intermediate steps, leading to a more holistic approach to traffic optimization.
- **Efficient Learning**: Utilizing Q-Learning, our agents learned optimal policies efficiently.

Chapter 4

Reformulation of the Problem as an RL System: Multi-Agent Approach

4.1 Introduction

Previously, we explored a centralized approach to optimizing traffic flow on two routes between an origin and a destination. The model employed a central agent that made decisions to allocate traffic to two routes: a and b. **We successfully demonstrated how Reinforcement Learning (RL) could be used to find the System Optimum (SO) and User Equilibrium (UE) in such system.**

Building on that success, the objective of this chapter is to dive deeper by employing a **decentralized approach**. Instead of having a single central agent managing the entire traffic, we aim to model **each individual traveler as a separate agent**. Each of these agents will use RL to make their own decisions about which route to take.

The decentralized model brings us closer to real-world scenarios where each traveler makes their own choice based on personal preferences and real-time conditions. It introduces an extra layer of complexity and realism that was not present in the centralized model.

The central questions are:

- Will a decentralized system of agents learn to achieve the System Optimum (SO) and User Equilibrium (UE)?
- How do individual agent behaviors evolve over time?
- How does the decentralized approach compare with the centralized one?

4.2 Remodeling the System as a RL Problem

4.2.1 Environment

Our environment has two routes, A and B, between an origin O and a destination D.

- **State Space**: The state is a tuple (K, x, y), where K is the current location of the vehicle, and x and y are the numbers of vehicles on routes a and b, respectively. K can take one of the following values: s, a, b
- **Action Space**: Binary: either choose route a or route b.

4.2.2 Agents

- Agents decide which route to take to maximize the reward provided by the environment.
- Q-learning enables the agents to learn from the rewards they get from their actions.

4.2.3 Rewards

- **For System Optimum (SO)**: Minimize the total system travel time.
- **For User Equilibrium (UE)**: Make travel times on both routes equal.

The same trick of densening this reward for each step, yet putting more emphasis on the final reward is applied here as well. **Each driver is penalized/rewarded more drastically as the number of vehicles in the traffic increases**. For details, see the previous section.

4.2.4 States and Actions

- **Initial State**: $(s, 0, 0)$ where s denotes the initial state, and the zeros represent that initially no vehicles are on either route.
- **Actions**: Agents choose between routes a or b.
- Each agent takes only one action in an episode, choosing one of the routes. They all start from the state (s, x, y), where x, y is the number of vehicles on both routes at that given moment. Then they transition to (k, x, y), where k is the route they picked and x, y is the congestion after their decision.

4.2.5 Why Q-learning?

- Model-free: Does not require an understanding of the dynamics of the environment.
- Online learning: Can learn while interacting with the environment.

- Learning the qualities of actions for given states is much more effective for this problem.

4.2.6 Design Decisions for Realistic Modeling

- **Agents take decisions in each episode in the same order**: In real life, every morning people usually use the city roads more or less in the same order. How early heading to your workplace depends on your shift hours, which for this simulation we can assume to be constant every day. This will speed up the convergence, but one can shuffle the order of agents in each episode and conduct the same learning with many more episodes.
- **Agents know the congestion before making a decision**: In real life, if a driver wants to make a decision regarding the route selection based on something, they usually use map applications or their knowledge about usual congestion on roads. Given their knowledge, though, they still have their own independent decision-making processes. Here, we modeled the system in a way that it is closer to the scenario in which every driver checks traffic congestion from an online source before heading to the destination. It can be also seen as every agent is the navigation system that drivers use, which uses real-time congestion data for giving directions.
- **Epsilon-learning**: Each agent occasionally makes random decisions during the learning phase. This way they understand better about the environment dynamics and the consequences of different actions. Epsilon is decayed throughout the training so that the agents learn how to reach to system optimal by coordinating with each other.

4.3 Computational Complexity

4.3.1 Environment Complexity

- step method: $O(1)$
- calculate_reward: $O(1)$

4.3.2 Agent Complexity

- get_action: $O(1)$ (remember: there are only two possible actions)
- update_Q_values: $O(1)$
- decay_epsilon: $O(1)$

4.3.3 Training Loop Complexity

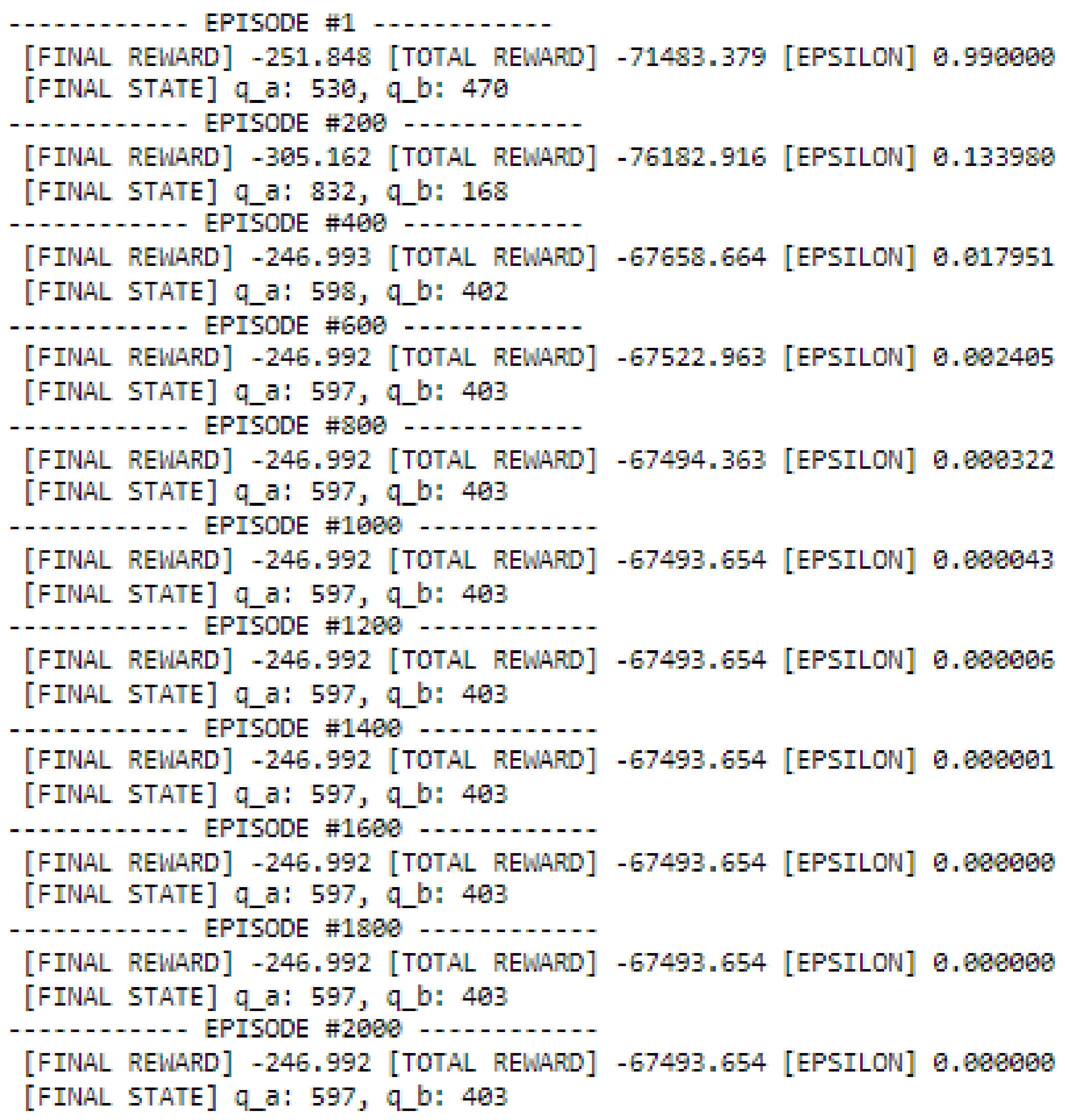
- Outer loop (num_episodes): $O(N)$, where N is the number of episodes.

- Inner loop (per agent): $O(Q)$, where Q is the number of agents.

Therefore, the overall complexity of the implementation is $O(N \times Q)$.

4.4 SO: Training and Stats

The training is configured with 1000 episodes, with $epsilon = 1.0$. Epsilon is decayed for each agent with a multiplier of 0.99 at the end of each episode.



```
------------ EPISODE #1 ------------
 [FINAL REWARD] -251.848 [TOTAL REWARD] -71483.379 [EPSILON] 0.990000
 [FINAL STATE] q_a: 530, q_b: 470
------------ EPISODE #200 ------------
 [FINAL REWARD] -305.162 [TOTAL REWARD] -76182.916 [EPSILON] 0.133980
 [FINAL STATE] q_a: 832, q_b: 168
------------ EPISODE #400 ------------
 [FINAL REWARD] -246.993 [TOTAL REWARD] -67658.664 [EPSILON] 0.017951
 [FINAL STATE] q_a: 598, q_b: 402
------------ EPISODE #600 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -67522.963 [EPSILON] 0.002405
 [FINAL STATE] q_a: 597, q_b: 403
------------ EPISODE #800 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -67494.363 [EPSILON] 0.000322
 [FINAL STATE] q_a: 597, q_b: 403
------------ EPISODE #1000 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -67493.654 [EPSILON] 0.000043
 [FINAL STATE] q_a: 597, q_b: 403
------------ EPISODE #1200 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -67493.654 [EPSILON] 0.000006
 [FINAL STATE] q_a: 597, q_b: 403
------------ EPISODE #1400 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -67493.654 [EPSILON] 0.000001
 [FINAL STATE] q_a: 597, q_b: 403
------------ EPISODE #1600 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -67493.654 [EPSILON] 0.000000
 [FINAL STATE] q_a: 597, q_b: 403
------------ EPISODE #1800 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -67493.654 [EPSILON] 0.000000
 [FINAL STATE] q_a: 597, q_b: 403
------------ EPISODE #2000 ------------
 [FINAL REWARD] -246.992 [TOTAL REWARD] -67493.654 [EPSILON] 0.000000
 [FINAL STATE] q_a: 597, q_b: 403
```

Figure 4.1: Training logs

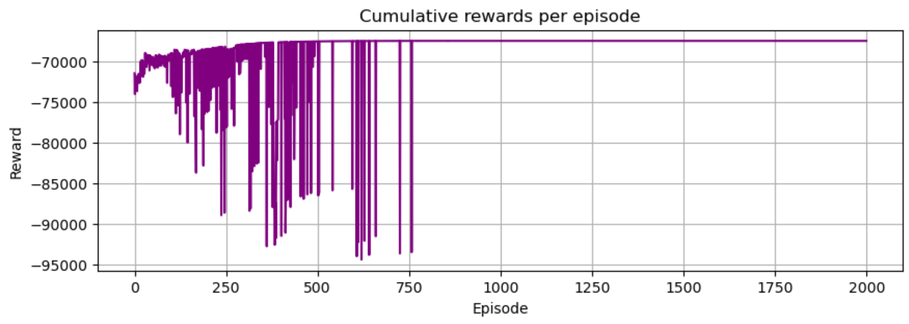


Figure 4.2: Episode rewards collected throughout the training

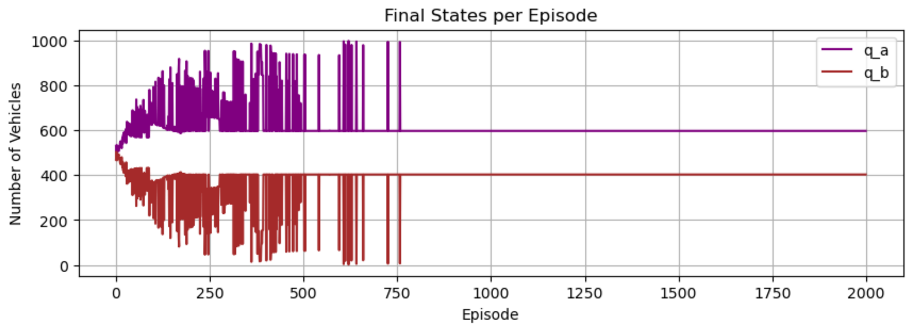


Figure 4.3: Final states at each episode

Figures-4.1, 4.2 and 4.3 show that the training successfully converges to the correct solution. The trend followed throughout the training also shows the efficiency of the training.

4.5 SO: Testing the Agents

We test our agents for one episode with complete deterministic decision-making.

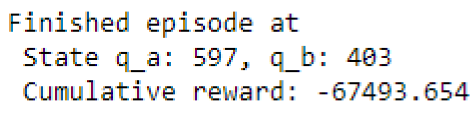

```
Finished episode at
 State q_a: 597, q_b: 403
 Cumulative reward: -67493.654
```

Figure 4.4: Final state at the end of the episode

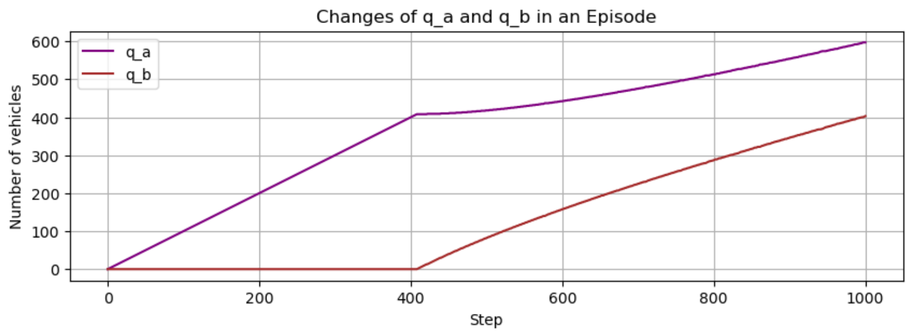


Figure 4.5: q_a and q_b throughout the episode.

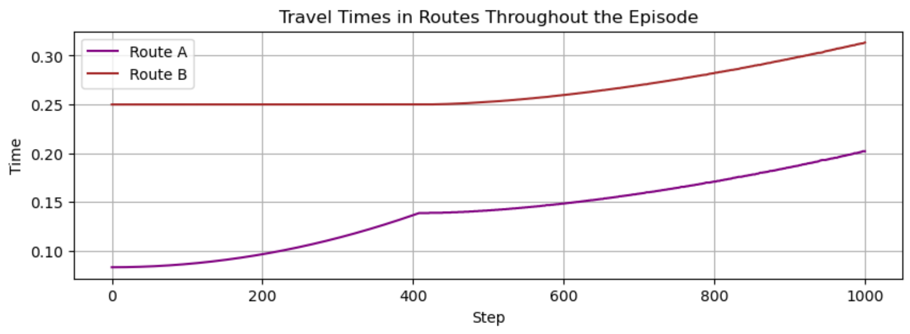


Figure 4.6: t_a and t_b throughout the episode.

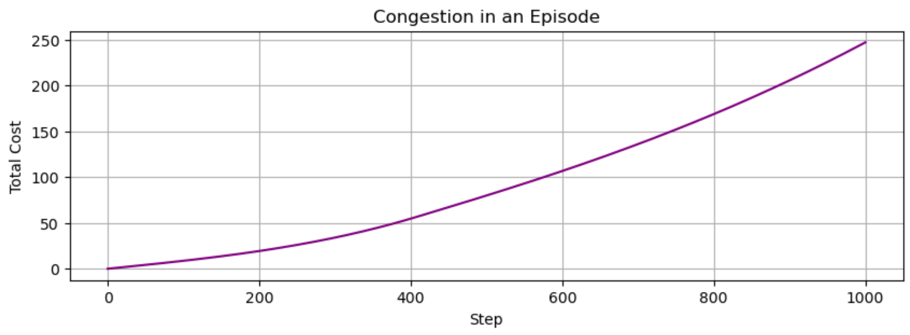
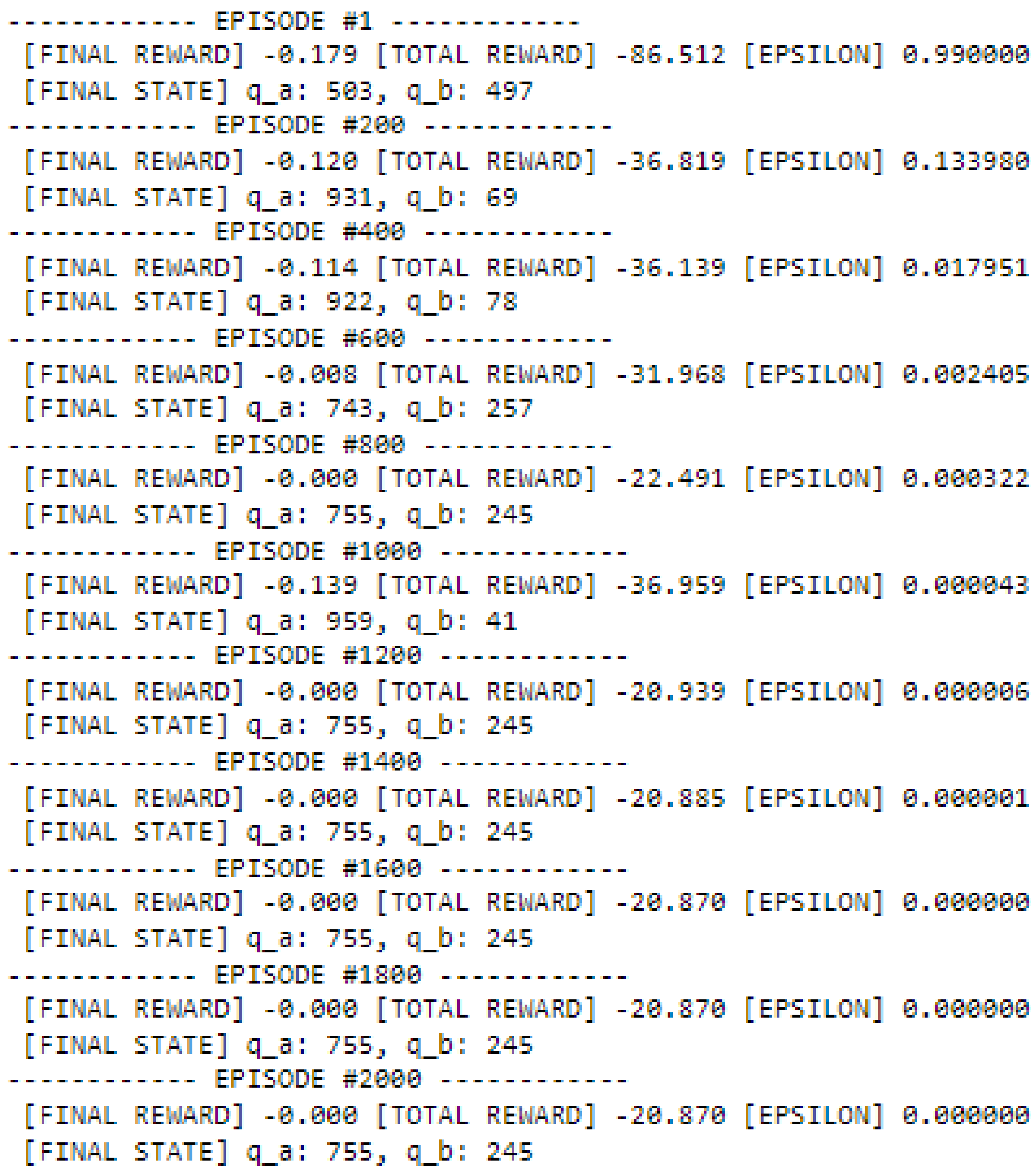


Figure 4.7: Total cost throughout the episode.

Our agents find the SO solution at the end of the episode.

4.6 UE: Training and Stats

The training is configured with 1000 episodes, with $epsilon = 1.0$. Epsilon is decayed for each agent with a multiplier of 0.99 at the end of each episode.



```
------------ EPISODE #1 ------------
 [FINAL REWARD] -0.179 [TOTAL REWARD] -86.512 [EPSILON] 0.990000
 [FINAL STATE] q_a: 503, q_b: 497
------------ EPISODE #200 ------------
 [FINAL REWARD] -0.120 [TOTAL REWARD] -36.819 [EPSILON] 0.133980
 [FINAL STATE] q_a: 931, q_b: 69
------------ EPISODE #400 ------------
 [FINAL REWARD] -0.114 [TOTAL REWARD] -36.139 [EPSILON] 0.017951
 [FINAL STATE] q_a: 922, q_b: 78
------------ EPISODE #600 ------------
 [FINAL REWARD] -0.008 [TOTAL REWARD] -31.968 [EPSILON] 0.002405
 [FINAL STATE] q_a: 743, q_b: 257
------------ EPISODE #800 ------------
 [FINAL REWARD] -0.000 [TOTAL REWARD] -22.491 [EPSILON] 0.000322
 [FINAL STATE] q_a: 755, q_b: 245
------------ EPISODE #1000 ------------
 [FINAL REWARD] -0.139 [TOTAL REWARD] -36.959 [EPSILON] 0.000043
 [FINAL STATE] q_a: 959, q_b: 41
------------ EPISODE #1200 ------------
 [FINAL REWARD] -0.000 [TOTAL REWARD] -20.939 [EPSILON] 0.000006
 [FINAL STATE] q_a: 755, q_b: 245
------------ EPISODE #1400 ------------
 [FINAL REWARD] -0.000 [TOTAL REWARD] -20.885 [EPSILON] 0.000001
 [FINAL STATE] q_a: 755, q_b: 245
------------ EPISODE #1600 ------------
 [FINAL REWARD] -0.000 [TOTAL REWARD] -20.870 [EPSILON] 0.000000
 [FINAL STATE] q_a: 755, q_b: 245
------------ EPISODE #1800 ------------
 [FINAL REWARD] -0.000 [TOTAL REWARD] -20.870 [EPSILON] 0.000000
 [FINAL STATE] q_a: 755, q_b: 245
------------ EPISODE #2000 ------------
 [FINAL REWARD] -0.000 [TOTAL REWARD] -20.870 [EPSILON] 0.000000
 [FINAL STATE] q_a: 755, q_b: 245
```

Figure 4.8: Training logs

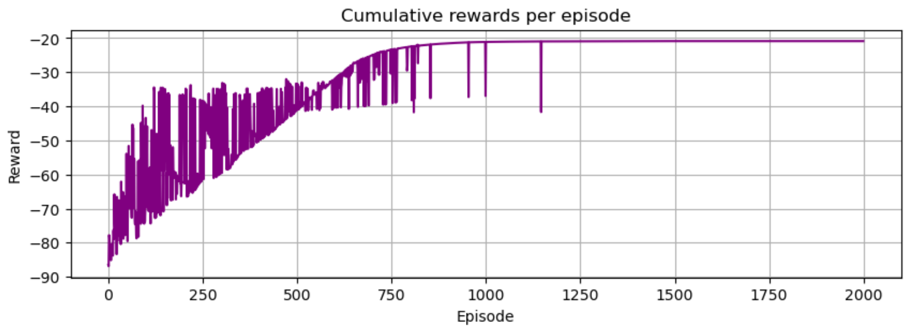


Figure 4.9: Episode rewards collected throughout the training

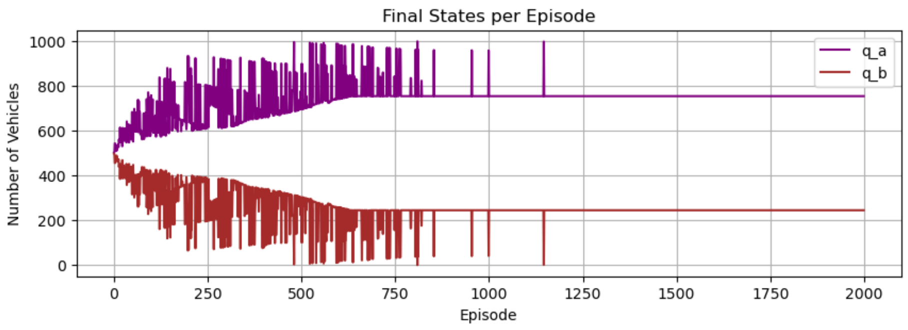
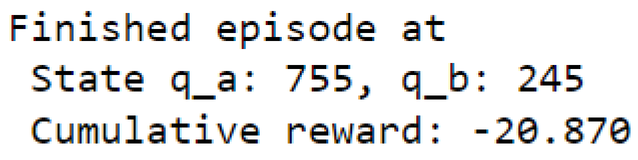


Figure 4.10: Final states at each episode

Figures-4.8, 4.9 and 4.10 show that the training successfully converges to the correct solution. The trend followed throughout the training also shows the efficiency of the training.

4.7 UE: Testing the Agents

We test our agents for one episode with complete deterministic decision-making.



```
Finished episode at
 State q_a: 755, q_b: 245
 Cumulative reward: -20.870
```

Figure 4.11: Final state at the end of the episode

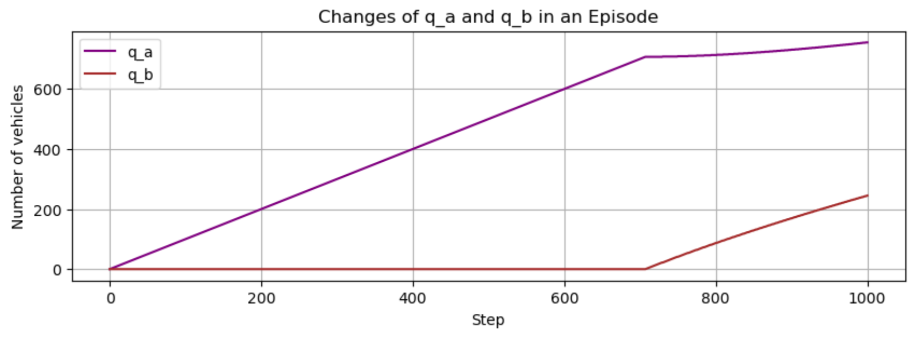


Figure 4.12: q_a and q_b throughout the episode.

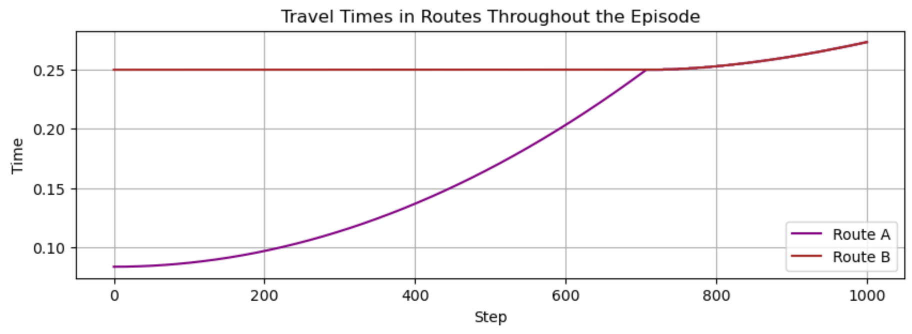


Figure 4.13: t_a and t_b throughout the episode.

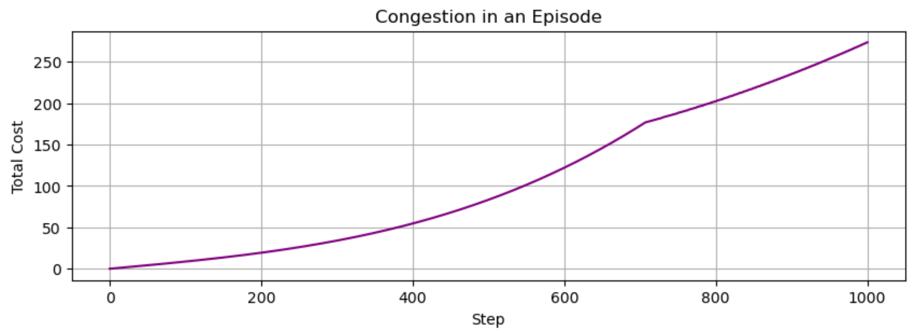


Figure 4.14: Total cost throughout the episode.

Our agents find the UE solution at the end of the episode. An important remark is, **all drivers chose Route A until the travel times of both routes are equalized. This can be seen as all drivers chose the shortcut route until it became too congested that it did not make a difference anymore. After the travel times equalized, they decided to cooperatively divide between two routes. This is something we did not observe in the centralized approach!**

4.8 Conclusion

In this chapter, we dealt with the problem of traffic optimization from multiple perspectives, employing a decentralized (multi-agent) model after the centralized one in the previous part of the project. We utilized Q-learning as our go-to reinforcement learning algorithm, demonstrating its adaptability in both settings.

Through careful design of state, action, and reward spaces, we were able to achieve System Optimum (SO) and User Equilibrium (UE) states. Moreover, the multi-agent model enabled us to examine individual behaviors and see some behaviors that we did not see in the centralized approach.

In the end, we have created a community that makes logical decisions and intelligently cooperates to reach SO and UE.

Chapter 5

Deep RL Based Urban Mobility Optimization

5.1 Introduction

In this part of the work, the system is slightly modified so it will be ideal for a solution using deep reinforcement learning techniques. Previously, we have seen how the initial problem can be modeled and resolved as an RL problem, and this time, we will go one step further by increasing the system complexity and demonstrate a more sophisticated solution. We will see under added complexities, if a deep RL model can provide any improvements.

5.2 System Description

5.2.1 Added Complexity

The environment now consists of 3 routes, connecting the origin to the destination. Also, we have a randomness factor to it, which increases the congestion by a little bit occasionally.

5.2.2 Episode

An episode consists set number of steps, each of them is a vehicle choosing a route. The episode is finalized when all vehicles are on a route.

5.2.3 Environment

The environment is modeled as a traffic system consisting of three routes (A, B, C) connecting an origin to a destination. Each route has a different capacity and free-flow speed:

- **Route A**: Capacity = 20, Free-flow travel time = 5
- **Route B**: Capacity = 25, Free-flow travel time = 8
- **Route C**: Capacity = 40, Free-flow travel time = 11

The system operates with a set of vehicles and introduces stochasticity ($randomness = 0.05$) to represent real-world uncertainties such as weather conditions. Number of vehicles is set to 50 for this implementation. For now, we focused solely on whether our implementation could offer any improvements to the initial case. One can simply increase the number of vehicles and rerun the code for a more realistic scenario, at the expense of more training time.

5.2.4 Problem as a RL Task

The system can be seen as the playground of a central traffic management agent, or a set of agents trying to get to their destination as fast as possible, sharing common knowledge. At each step, the agent is responsible for selecting one of the three routes to reach its destination.

- The **state** space is a vector comprising the current traffic on each route and the occurrence of random events.
- The **action** space contains three discrete actions corresponding to the choice of the route. It is selected from $0, 1, 2$, corresponding to routes A, B, or C respectively.
- The **reward function** is designed to penalize longer travel times while balancing the load among the routes. At each step for the given route selection x, the reward is $t_x(q_x)$ on that route where q_x is the current traffic. Thanks to this, it is aimed to enable the agent to pick the fastest route at each step. In the code, there is also a jackpot reward disabled, which one can enable back, which rewards if the travel times on all routes are equalized.

5.2.5 Algorithms and Methods

We employ a **Deep Q-Network (DQN)** with **experience replay** for training each agent. Key hyper-parameters are:

- $batch\_size = 8$: Number of experiences used in each training iteration
- $num\_episodes = 500$: Total number of episodes for training
- $gamma = 0.8$: Discount factor

- $epsilon = 1.0$: Initial exploration rate
- $epsilon\_decay = 0.99$: Epsilon decay applied at each episode

The neural network architecture consists of two hidden layers and employs Mean Squared Error (MSE) as the loss function. As the name of the method suggests, **the network yields the Q-value estimations of each possible action given the state**.

5.3 Complexity

The primary computational complexities stem from:

- **Neural Network Training**: The DQN agent uses a fully connected neural network with two hidden layers, making the training complexity $O(n^2)$ for n neurons.
- **Experience Replay**: The complexity for experience replay is $O(n \times m)$ for n experiences and m batch size.
- **Episodes and Actions**: The total number of actions performed is $num\_episodes \times num\_vehicles$.

5.3.1 Space Complexity

- **Experience Memory**: The agent stores state, action, reward, and next_state, each of which is a vector or scalar. Given that we store these values for each vehicle for every episode, for n experiences, the space complexity is $O(4 \times num\_vehicles \times num\_episodes)$.
- **Q-Table Approximation**: The neural network itself is a function approximator for the Q-table. With n_1 and n_2 neurons in the hidden layers, the space required to store the weights and biases is $O(n_1 + n_1 \times n_2 + n_2)$.

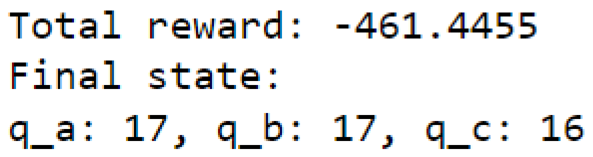
5.4 Results

The agent successfully navigated the environment with increasing cumulative rewards over episodes, thus demonstrating the ability to learn optimal policies for traffic management. Results are illustrated with rewards and final states at regular intervals.

As one can see, the training still seems to be in an unstable trend in the end. This can be because of the ongoing convergence, or an effect of the stochastic nature of the environment. Either way, one can explore where this training goes if given more time.

5.5 Random Agent

To have a better understanding of the improvements after the training, we will first have a look at the performance of the initial agent, which picks actions completely at random.



```
Total reward: -461.4455
Final state:
q_a: 17, q_b: 17, q_c: 16
```

Figure 5.1: Final state after one episode with the random agent

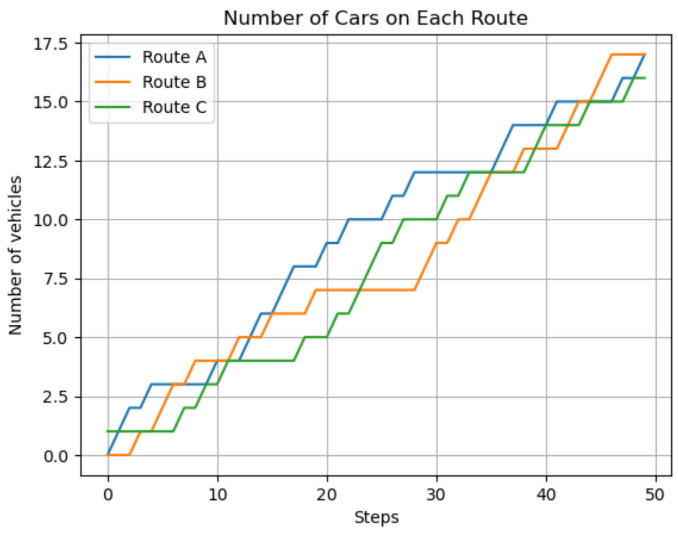


Figure 5.2: Vehicle assignments at every step in an episode

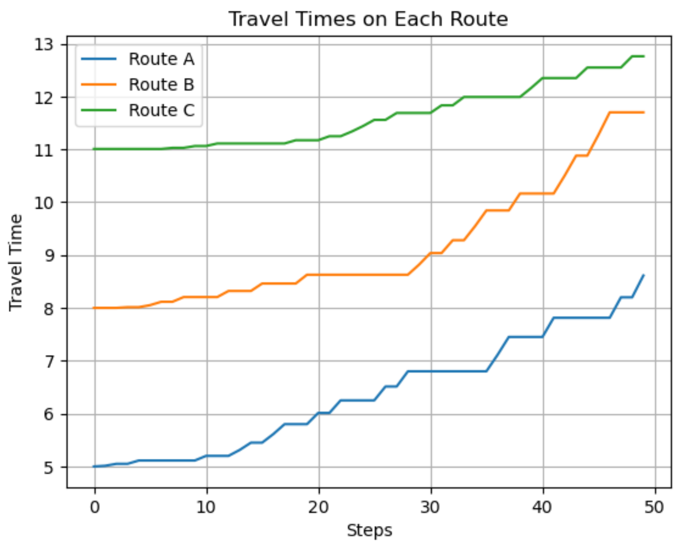
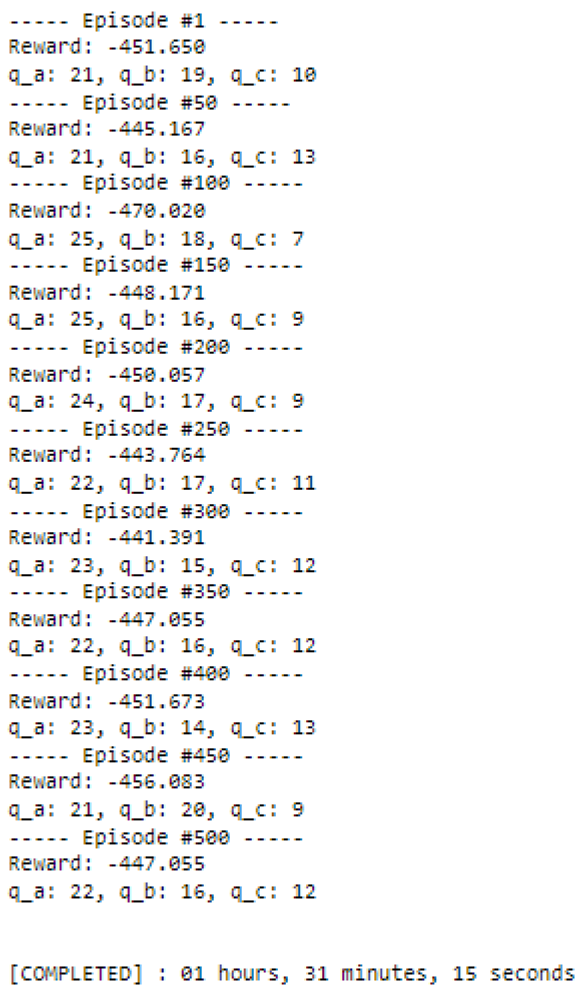


Figure 5.3: Change of congestion on routes throughout the episode

5.6 Training Stats

We have trained the agent for 500 episodes, with the initial $epsilon = 1.0$, and decaying it at each episode with the multiplier 0.99.



```
----- Episode #1 -----
Reward: -451.650
q_a: 21, q_b: 19, q_c: 10
----- Episode #50 -----
Reward: -445.167
q_a: 21, q_b: 16, q_c: 13
----- Episode #100 -----
Reward: -470.020
q_a: 25, q_b: 18, q_c: 7
----- Episode #150 -----
Reward: -448.171
q_a: 25, q_b: 16, q_c: 9
----- Episode #200 -----
Reward: -450.057
q_a: 24, q_b: 17, q_c: 9
----- Episode #250 -----
Reward: -443.764
q_a: 22, q_b: 17, q_c: 11
----- Episode #300 -----
Reward: -441.391
q_a: 23, q_b: 15, q_c: 12
----- Episode #350 -----
Reward: -447.055
q_a: 22, q_b: 16, q_c: 12
----- Episode #400 -----
Reward: -451.673
q_a: 23, q_b: 14, q_c: 13
----- Episode #450 -----
Reward: -456.083
q_a: 21, q_b: 20, q_c: 9
----- Episode #500 -----
Reward: -447.055
q_a: 22, q_b: 16, q_c: 12


[COMPLETED] : 01 hours, 31 minutes, 15 seconds
```

Figure 5.4: Training logs

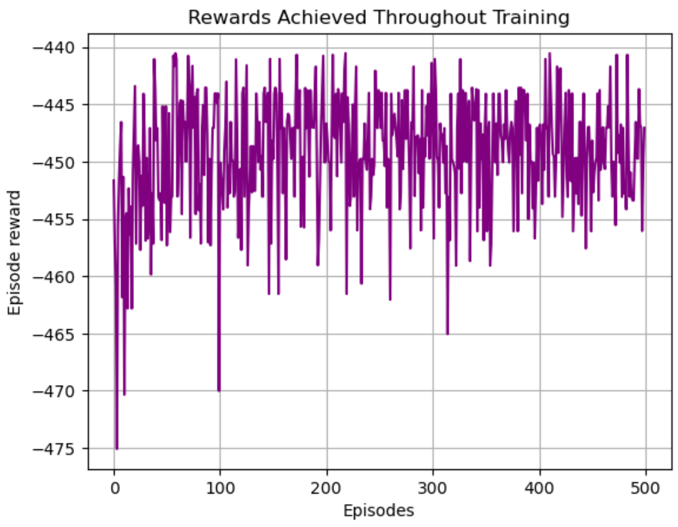


Figure 5.5: Rewards collected at each episode throughout the training

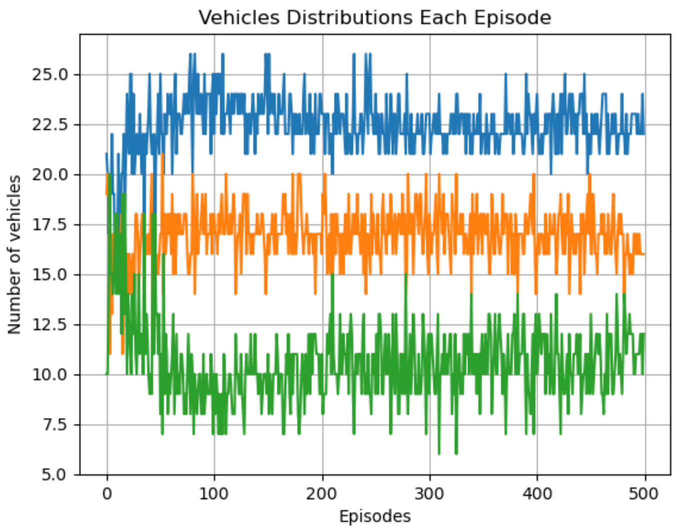
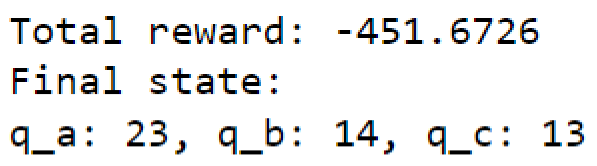


Figure 5.6: Final states at each episode throughout the training

5.7 Testing the Agent

Now we show the performance of the trained agent in our environment. It will start acting in a fresh environment and we will observe at which state it will terminate. We will also note its decision-making at each step.



```
Total reward: -451.6726
Final state:
q_a: 23, q_b: 14, q_c: 13
```

Figure 5.7: Final state after one episode with the trained agent

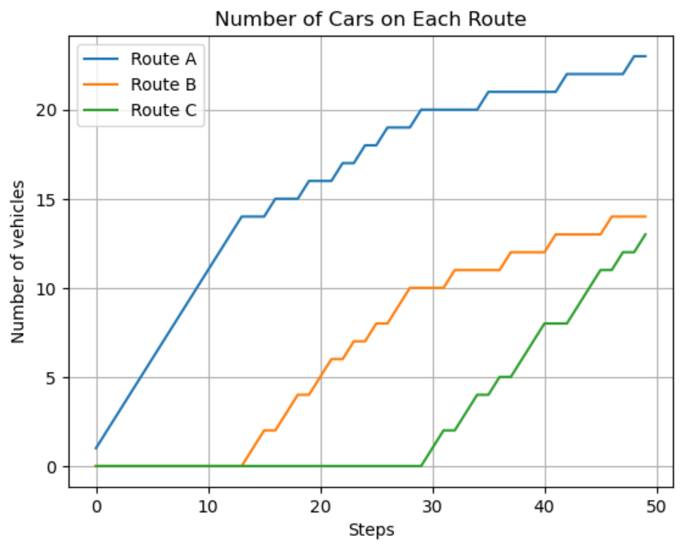


Figure 5.8: Vehicle assignments at every step in an episode

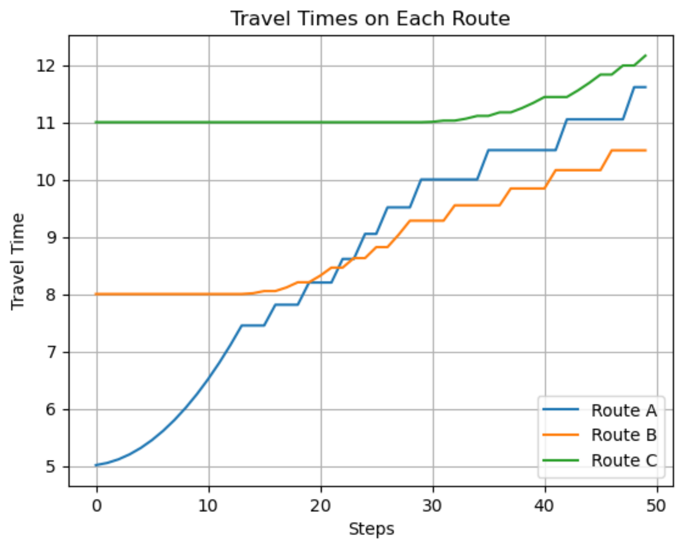


Figure 5.9: Change of congestion on routes throughout the episode

We see that the agent does not assign more vehicles to the longer routes until the faster one becomes congested enough. This shows a similar thought process to the multi-agent RL system described in the previous section.

5.8 Conclusion

We have designed a new system with a few added complexities and tried to optimize it using a more sophisticated RL algorithm, Deep-Q-Learning. We tried our best to make the implementation as efficient as possible, even trying to utilize GPU resources in the replay function, but still, we observed a huge time consumption increase compared to the previous sections. However, we still observe that our agent gains the understanding of which route offers a more express travel to the destination, and at which configuration this changes. One can explore further with a different network architecture and possibly more training episodes.

Chapter 6

Problem Extension: Driveington

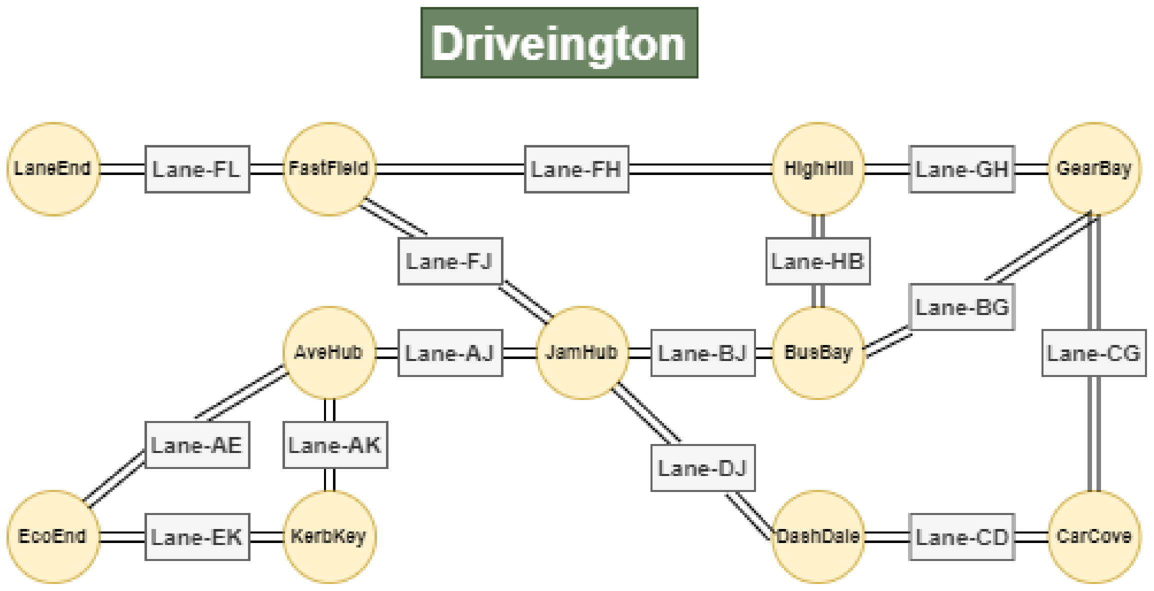


Figure 6.1: Map of Driveington

6.1 Introduction

In this chapter, the focus is on how to extend the traffic optimization problem in hand, to a system that relates better to real city traffic management.

We have designed the above traffic network, which belongs to an imaginary city called **Driveington**.

- In this city, the drivers start driving from their origin to their destinations every morning. Each agent has their own origin and destination pairs. These pairs are generated randomly based on a sampling table created by assigning probabilities that indicate for a given city section, how likely it is to be a destination or an origin. *For example, the section "LaneEnd", thought as a suburban area, is much more likely to be an origin for a driver than a destination. On the other hand, the section "JamHub" is considered as the city center, where living costs are high but a popular destination.*

- We have employed an object-oriented approach, which distinguishes the characteristics of each entity in a traffic network.

- From an RL perspective, each vehicle (driver) is an agent, and the city itself is the environment. Each vehicle can be one of two types: a bus or a car. The difference is that a bus has twice as much influence on the congestion on a road than a car.

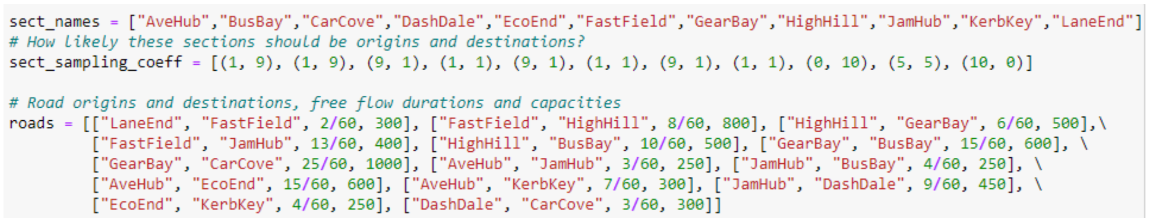
- The training is very time-consuming, therefore for now the selected population is 500. We also implemented stochastic behaviors on the roads, vehicles, and the cities. But for the same time constraints, they are disabled for this implementation. If activated, a vehicle can change its destination randomly, a road can become much more congested permanently, and the city might not allow a driver to transfer to a new road on a given time step. These are implemented by keeping in mind the real-life scenarios.

6.2 Traffic Network of Driveington as a RL Problem

6.2.1 Environment

The city itself acts as an RL environment. It takes the next section ID as an action and returns the corresponding state with a step reward. If the driver has arrived at its destination, any action submitted by the agent is treated the same and movement is not allowed.

The map of the Driveington, with its sections and the roads connecting them, is given in Figure-6.1. The origin/destination sampling coefficients and the road parameters are given in Figure-6.2.



```
sect_names = ["AveHub","BusBay","CarCove","DashDale","EcoEnd","FastField","GearBay","HighHill","JamHub","KerbKey","LaneEnd"]
# How likely these sections should be origins and destinations?
sect_sampling_coeff = [(1, 9), (1, 9), (9, 1), (1, 1), (9, 1), (1, 1), (9, 1), (1, 1), (0, 10), (5, 5), (10, 0)]

# Road origins and destinations, free flow durations and capacities
roads = [["LaneEnd", "FastField", 2/60, 300], ["FastField", "HighHill", 8/60, 800], ["HighHill", "GearBay", 6/60, 500],\
         ["FastField", "JamHub", 13/60, 400], ["HighHill", "BusBay", 10/60, 500], ["GearBay", "BusBay", 15/60, 600], \
         ["GearBay", "CarCove", 25/60, 1000], ["AveHub", "JamHub", 3/60, 250], ["JamHub", "BusBay", 4/60, 250], \
         ["AveHub", "EcoEnd", 15/60, 600], ["AveHub", "KerbKey", 7/60, 300], ["JamHub", "DashDale", 9/60, 450], \
         ["EcoEnd", "KerbKey", 4/60, 250], ["DashDale", "CarCove", 3/60, 300]]
```

Figure 6.2: Sections and roads of Driveington

6.2.2 State Space

The states in each step are tuples in the form *(origin, destination, a dictionary of possible roads and the number of vehicles using it)*. To make the training more efficient, the state space is made more discrete by applying a binning operation for the number of vehicles on each road. By using the number of vehicles instead of travel time for each road, it is aimed to develop an understanding for each agent about the characteristics of the different roads.

6.2.3 Actions

Given the state, the agent has access to the list of the next possible roads to take with their congestion. The selected action is the ID of the section that the selected route leads to. The IDs of each city section are their initials. The action selection policy is **deterministic**.

At the beginning of the training, the epsilon is equal to 1 for every agent. This is useful for exploring the state space, which is very large in this problem. After the epsilon decays down to 0, the agents start to exploit their policies.

6.2.4 Agents

Each vehicle is an agent. Therefore if the population is selected as 500, the system is a multi-agent system with 500 agents.

Each agent is from one of two types: bus or car. **A bus has twice as much influence on road congestion**. The buses are around 5% of the population.

Each agent is focused on reaching their destinations with maximal cumulative reward. To do that, each of them needs to pick routes that not only involve a minimal number of turns but also the time spent on each road should be minimal. Also considering that the agents take actions in an order, the decision of one agent affects another agent's decision. Considering all these, we can see that this is an example of a system with **competing agents**.

6.2.5 Rewards

The rewards are calculated as follows:

- $-t_x$ if the agent has not arrived, where x is the road agent picked and t_x is the current travel time on the road x.
- 0 if the agent has arrived at its destination, regardless of its action.

6.2.6 Stochastic Behaviors

For this implementation, the stochastic behaviors are disabled. If you wish to enable them, the entities have the following behaviors:

- An agent might change its destination in mid-trajectory.
- A road can get more congested temporarily, similar to real-life events like accidents or natural disasters.
- A city can reject the movement of an agent; the agent remains on the same road for that time step.

6.2.7 Episode

An episode is not constrained to a limited number of steps. An episode ends when all agents arrive at their destinations. Therefore, it might take as little as 1 step to thousands.

6.2.8 Learning

Given the above properties of this system, a Deep-Q-Learning approach might seem feasible. However, again, due to the time constraints and that we still observed improvements with this approach, we employed a Q-Learning technique. The Q-table is a look-up table that assigns each state-action pair an expected future reward.

6.3 Random Agent

Now we will have a look at the mobility when all the agents pick their actions completely at random. The results are obtained after one complete episode.

```
Finished episode in 140 steps.
Total reward: -1291.560
```

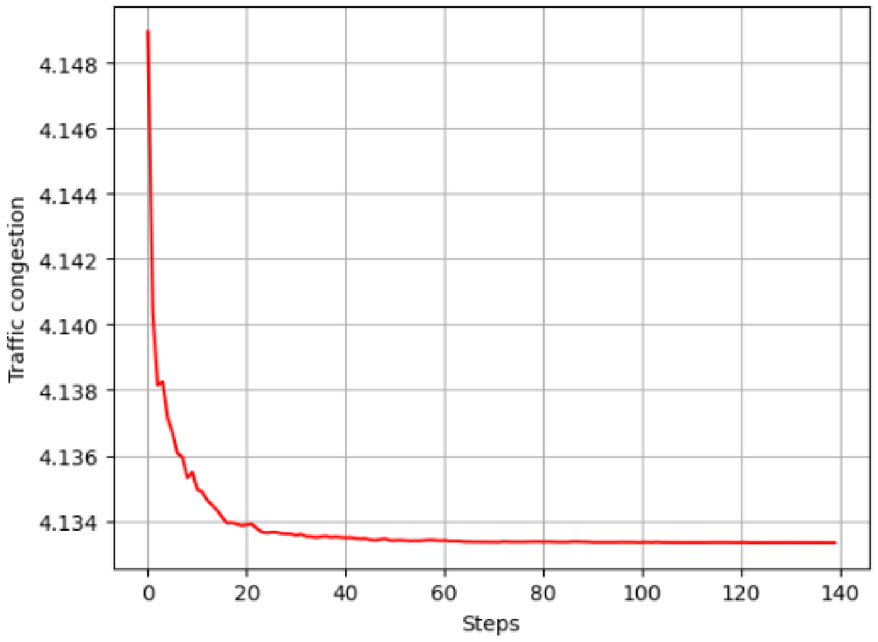
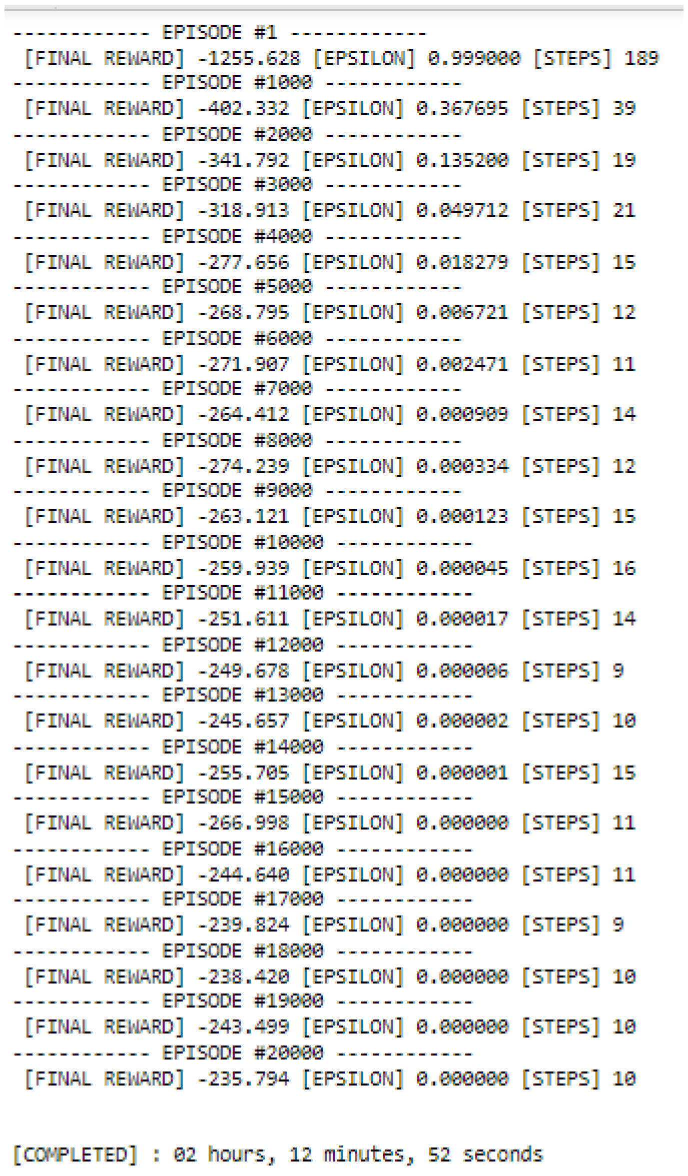


Figure 6.3: Mobility of the random agents in Driveington

6.4 Training Stats

We see that as the training progressed, we started to collect greater cumulative rewards, and the episodes got significantly shorter. This gives a clear indication of how agents started to understand the dynamics of the environment. Moreover, we see that $themaximumrewardisnotstrictlydependentonthenumberofsteps$, which shows that $theagentslearnthelowest-costpathsandnotthepathswiththeminimalnumberofturns$.



```
------------ EPISODE #1 ------------
 [FINAL REWARD] -1255.628 [EPSILON] 0.999000 [STEPS] 189
------------ EPISODE #1000 ------------
 [FINAL REWARD] -402.332 [EPSILON] 0.367695 [STEPS] 39
------------ EPISODE #2000 ------------
 [FINAL REWARD] -341.792 [EPSILON] 0.135200 [STEPS] 19
------------ EPISODE #3000 ------------
 [FINAL REWARD] -318.913 [EPSILON] 0.049712 [STEPS] 21
------------ EPISODE #4000 ------------
 [FINAL REWARD] -277.656 [EPSILON] 0.018279 [STEPS] 15
------------ EPISODE #5000 ------------
 [FINAL REWARD] -268.795 [EPSILON] 0.006721 [STEPS] 12
------------ EPISODE #6000 ------------
 [FINAL REWARD] -271.907 [EPSILON] 0.002471 [STEPS] 11
------------ EPISODE #7000 ------------
 [FINAL REWARD] -264.412 [EPSILON] 0.000909 [STEPS] 14
------------ EPISODE #8000 ------------
 [FINAL REWARD] -274.239 [EPSILON] 0.000334 [STEPS] 12
------------ EPISODE #9000 ------------
 [FINAL REWARD] -263.121 [EPSILON] 0.000123 [STEPS] 15
------------ EPISODE #10000 ------------
 [FINAL REWARD] -259.939 [EPSILON] 0.000045 [STEPS] 16
------------ EPISODE #11000 ------------
 [FINAL REWARD] -251.611 [EPSILON] 0.000017 [STEPS] 14
------------ EPISODE #12000 ------------
 [FINAL REWARD] -249.678 [EPSILON] 0.000006 [STEPS] 9
------------ EPISODE #13000 ------------
 [FINAL REWARD] -245.657 [EPSILON] 0.000002 [STEPS] 10
------------ EPISODE #14000 ------------
 [FINAL REWARD] -255.705 [EPSILON] 0.000001 [STEPS] 15
------------ EPISODE #15000 ------------
 [FINAL REWARD] -266.998 [EPSILON] 0.000000 [STEPS] 11
------------ EPISODE #16000 ------------
 [FINAL REWARD] -244.640 [EPSILON] 0.000000 [STEPS] 11
------------ EPISODE #17000 ------------
 [FINAL REWARD] -239.824 [EPSILON] 0.000000 [STEPS] 9
------------ EPISODE #18000 ------------
 [FINAL REWARD] -238.420 [EPSILON] 0.000000 [STEPS] 10
------------ EPISODE #19000 ------------
 [FINAL REWARD] -243.499 [EPSILON] 0.000000 [STEPS] 10
------------ EPISODE #20000 ------------
 [FINAL REWARD] -235.794 [EPSILON] 0.000000 [STEPS] 10


[COMPLETED] : 02 hours, 12 minutes, 52 seconds
```

Figure 6.4: Training logs

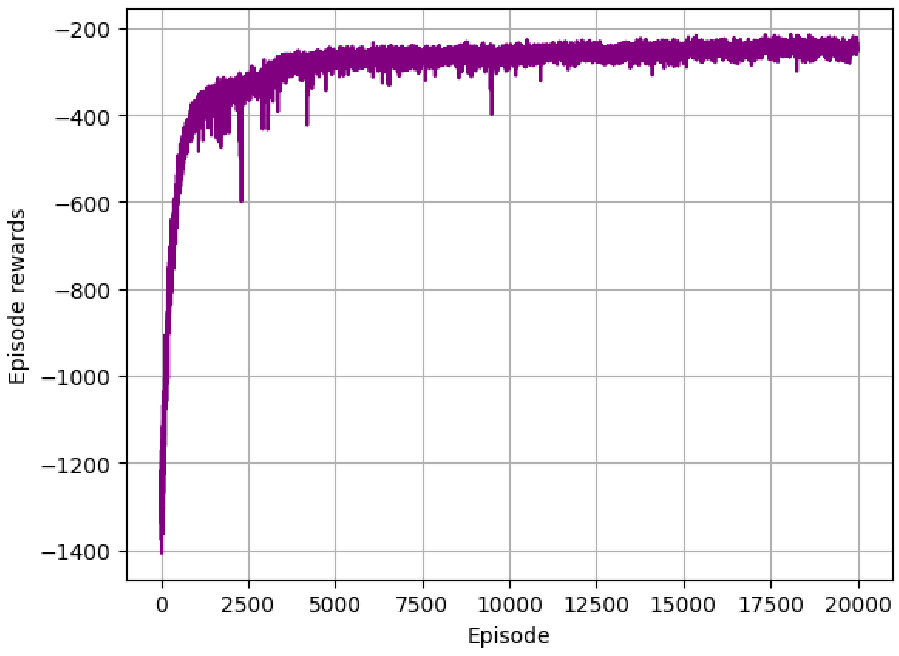


Figure 6.5: Episode rewards throughout the training

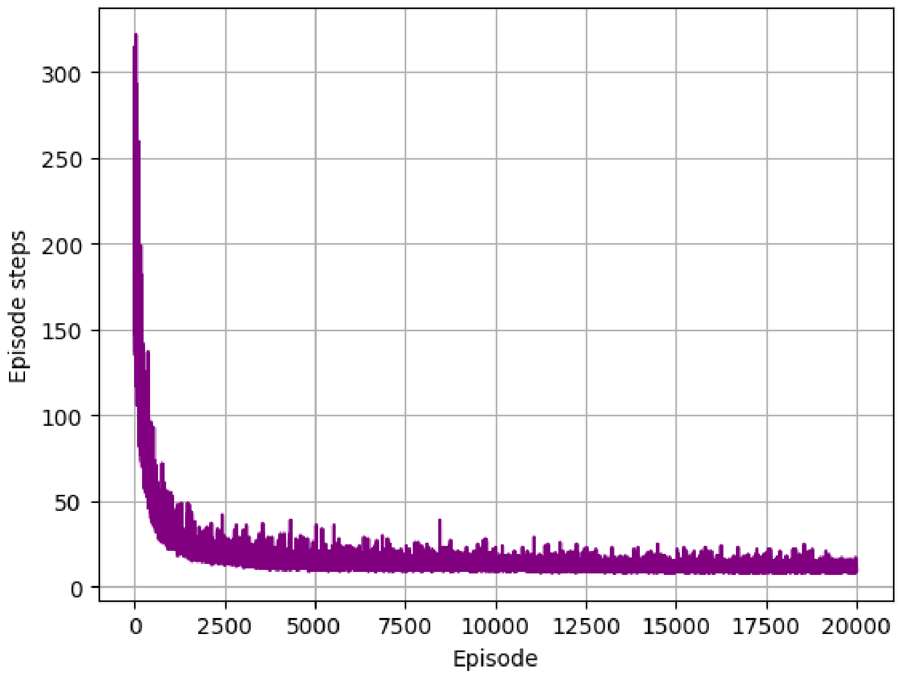
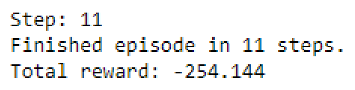


Figure 6.6: Length of episodes throughout the training

6.5 Testing the Agents

Now we have a look at the performance of our trained agents in our environment for one complete episode.



```
Step: 11
Finished episode in 11 steps.
Total reward: -254.144
```

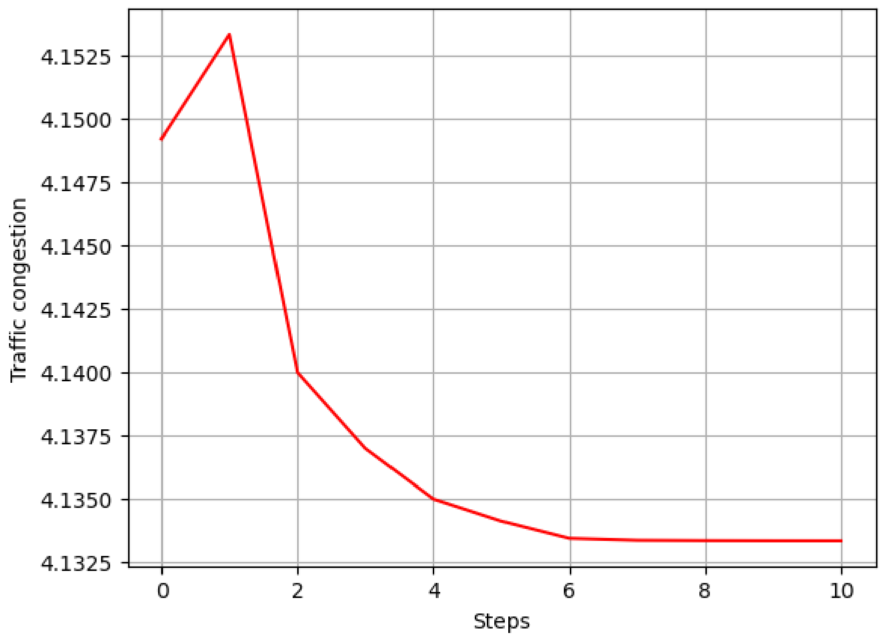


Figure 6.7: Mobility of the trained agents in Driveington

Chapter 7

Conclusions

This project aimed to explore and solve a transportation optimization problem, initially defined by a simple two-route setup, using both traditional analytical methods and RL-based techniques. The journey spanned from the analytical determination of System Optimum (SO) and User Equilibrium (UE) configurations to a Reinforcement Learning (RL) formulation that solved the problem from a centralized and decentralized perspective. We then elevated the problem complexity and transitioned to a Deep Q-learning approach. Finally, we conceptualized and tackled a new problem, 'Optimization of Traffic in Driveington,' simulating a cityscape with various routes and traffic conditions.

7.1 Significance

The work accomplished serves as a comprehensive guide for leveraging various optimization techniques in transportation. It demonstrates the versatility of RL frameworks in adapting to both simple and complex scenarios. In the case of 'Driveington', the RL-based approach showed its strength in managing high-dimensional state spaces, competing agents, and stochastic events, making the model highly applicable to real-world scenarios.

7.2 Complexity and Challenges

We discussed the algorithmic complexities associated with each method, highlighting that while basic RL-based solutions are computationally cheaper, they might lack the adaptability that more sophisticated RL methods bring. On the flip side, the deep RL-based solution requires extensive computational resources and time to train. However, it provides a solution that is applicable when dealing with high-dimensional and dynamic environments.

7.3 Statistical Insights

The performance metrics and visualizations underscored the efficiency of the various approaches. We tried to demonstrate not only the success achieved at the end of the training processes but also the overall efficiency of the training in the systematical convergence to a satisfactory solution.

7.4 Future Directions

Although this project covers a broad spectrum of techniques and complexities, there are several proposed directions for future work:

- Exploring other types of RL methods like Proximal Policy Optimization (PPO) for a more efficient learning process.
- In the case of deep learning-based methods, trying out different hyperparameters, other network architectures, and different software tricks.
- Extending the 'Driveington' model to consider various other types of agents, such as pedestrians and cyclists, for a more comprehensive traffic management system.

In summary, this project not only served as an academic exercise but also provided a solid starting point for someone who is willing to explore the potential of RL in urban mobility optimization. It is truly remarkable how the complexity can rapidly increase with a few changes in a traffic model, and that there is always a possible solution for it from the reinforcement learning framework.