# Discrete-Event Ride-Sharing Simulation: Final Project Report

## Introduction

The purpose of this project was to design and implement a ride-sharing simulator that models how drivers and riders interact in an urban network. The main goal was to explore how efficiently a fleet of cars can be dispatched to serve incoming rider requests, how long riders typically wait for service, and how effectively the fleet is utilized.  
  
In real-world systems like Uber and Lyft, the challenge is finding a balance between rider experience (short wait times, reliable service) and operational efficiency (maximizing how much drivers are actually on trips). This project aimed to answer key questions:  
- How well can the system handle requests under changing demand?  
- What are the trade-offs between fast pickup times and high driver utilization?  
- How evenly is work distributed across the fleet?  
  
By answering these questions through simulation, we can better understand the dynamics of ride-sharing systems and identify opportunities for optimization.

## Methodology

### Architecture

The simulator is built using a discrete-event engine. Instead of advancing in fixed time steps, it processes events (e.g., rider requests, pickups, dropoffs) in exact chronological order using Python’s heapq priority queue. This makes the simulation both efficient and realistic, since no time is wasted processing empty steps.

### Data Structures and Algorithms

- Graph Representation: The city is modeled as a weighted, undirected graph where nodes represent intersections and edges represent travel times. The graph is loaded from map.csv, which stores both edges and node coordinates.  
- Quadtree: Car locations are indexed in a quadtree, which makes finding the nearest available drivers much faster than scanning the entire fleet.  
- Dijkstra’s Algorithm: For routing, the simulator uses Dijkstra to compute the shortest path between a car’s location and a rider’s pickup or dropoff.  
- Dynamic Rider Generation: Instead of preloading requests, new riders are generated during the simulation using exponential inter-arrival times.  
- Event Handlers: Distinct event types (RIDER\_REQUEST, PICKUP\_ARRIVAL, DROPOFF\_ARRIVAL) are handled by dedicated methods.

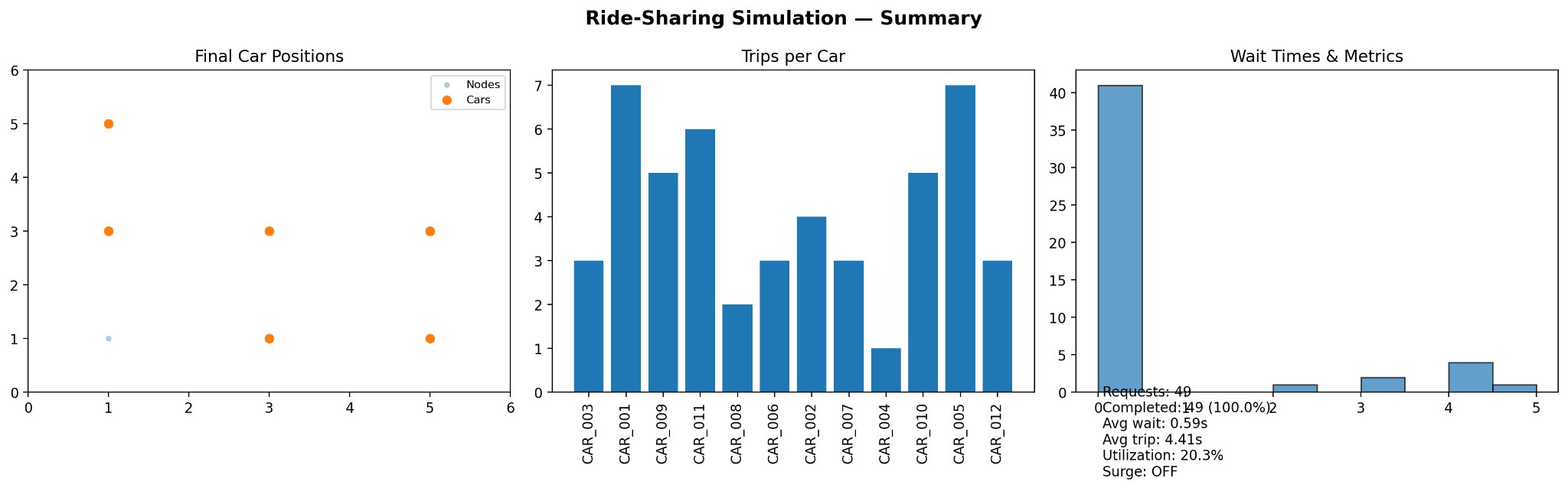
### Parameters for the Run

The analyzed simulation was run with the following setup:  
- Simulation Time: 100-time units  
- Fleet Size: 12 cars  
- Rider Requests: 49 generated dynamically  
- Arrival Rate: ~1 request every 2 time units  
- Surge Pricing: Disabled (focus on operational behavior)

## Results

### Visualization

The final integrated visualization (simulation\_summary.png) shows:  
1. Final Car Positions – where cars ended up after all requests were completed.  
2. Trips per Car – how many trips each vehicle completed.  
3. Wait Times & Metrics – a histogram of rider wait times and key performance indicators.



### Key Performance Indicators (KPIs)

|  |  |
| --- | --- |
| Total Rider Requests | 49 |
| Completed Trips | 49 (100%) |
| Average Wait Time | 0.59 time units |
| Average Trip Duration | 4.41 time units |
| Driver Utilization | 20.3% |
| Surge Pricing | OFF |

## Discussion

The results show an extremely efficient system from the rider’s perspective. All 49 requests were successfully matched, and the average wait time was under a single time unit (essentially instant service). The trip duration averaged just over 4 time units, reflecting reasonable network distances.  
  
However, this came at the cost of low driver utilization (20.3%). This means cars spent much of the simulation idle, waiting for requests. While this kept wait times low, it is not sustainable in real-world systems where driver hours are expensive.  
  
The workload distribution also highlights imbalance. Some cars (CAR\_001, CAR\_005) completed 7 trips each, while others like CAR\_004 completed only 1. This suggests that demand was unevenly distributed across the network and cars located closer to high-demand areas naturally received more trips.  
  
These results reveal the classic trade-off in ride-sharing:  
- High Service Quality → Low wait times, but low utilization.  
- High Utilization → More efficient fleet, but longer wait times.

Limitations:  
- No traffic or congestion effects.  
- No driver preferences or shifts.  
- No surge pricing or dynamic fares.  
- Only single-passenger, point-to-point trips (no pooling).

Hypothesis for Improvement: If the fleet size were reduced from 12 to 8 cars, rider wait times would likely increase to 1.5–2.0 units, but utilization could rise to 35–40%. This would better balance efficiency and service quality.

## Conclusion

This project successfully built a working ride-sharing simulator that integrates an event-driven engine, quadtree indexing, and Dijkstra pathfinding. The results demonstrated excellent rider experience but highlighted inefficiencies in fleet utilization.  
  
Future improvements could include adding traffic modeling, experimenting with dynamic fleet sizing, or enabling surge pricing. These features would make the simulator even more realistic and provide deeper insights into how ride-sharing platforms operate in real cities.