Computer Aided Simulation Lab

Simulation of a Car Sharing System

Labs 2 and 3

Riccardo Bracciale
S338616

Lab 2 Assignment:

The purpose of this lab is to develop a simulation model that captures the dynamics of a car-sharing system.
In this system:

- Users appear at various random locations and, using a smartphone app, identify and reserve an available car nearby.
- After reserving, users can drive the car to their destination and leave it either in designated parking spots (e.g. rechargeable stations) or any available space.
- Cars may be partially relocated to better align with demand when necessary.

Introduction

The purpose of this laboratory is to develop a simulation model that captures the operational dynamics of a **car-sharing system** as described above.

The following sections identify the fundamental system parameters, define the main performance indicators, outline the conceptual model design, and describe the data structures required to support the simulation.

1 Model Design

Key Design Principle: The simulation model is based on **discrete-event simulation** principles, where the system evolves through a sequence of discrete events rather than continuous time steps.

The whole system is represented as a collection of **entities** (users, vehicles, charging stations) that interact over time through a series of **events** (user arrivals, vehicle reservations, trip completions, charging operations, and relocations).

1.1 Entities

The main entities of the car-sharing system are as follows:

- **User**: contains information such as current location, desired destination, reservation status, and time of arrival.
- **Vehicle**: stores attributes including current position, battery level, state (available, in use, charging, or reserved), and time since last relocation.
- **Car Relocator**: a special worker responsible for relocating low-battery vehicles to charging stations.

1.2 Entities Behavior

Users are expected to sign up for the platform: in the simulation, they can be created all instantly at the beginning of the simulation or arrive over time according to a stochastic process.

Once registered, users will randomly make requests to rent a car (following a stochastic process), with random destinations inside the service area.

All the entities interact according to the following processes:

Reservation Process:

1. User makes a car request at a random location
2. System searches for available cars within a search radius
3. User is assigned to the *closest available car*, if no cars are available, user retries (up to a maximum number of attempts)
4. After exhausting attempts, user abandons the request

Rental Process:

1. After a successful reservation, the user will get to the car's location after a certain walking time, which depends on the distance between the user's initial position and the car's position.

2. Then the user drives the car to the desired destination, the time taken will depend on the **traffic conditions** between the two points, which can change over time.

3. During the trip, the car uses up energy based on the distance driven.

4. If a car's battery is below a certain level at the end of a rental, it will be taken to be charged by a special worker called a *relocator*. This person will pick up the car and drive it to the nearest charging station.

These processes are implemented through the events described in the following section.

1.3 Events

The simulation operates through a series of discrete events that represent key moments in the system's operation:

- **User Subscription Event**: A new user joins the car-sharing platform and becomes eligible to make reservations. This event triggers the scheduling of the user's first reservation attempt.
- **Reservation Event**: A user attempts to find and reserve an available car near their current location. The system searches for the nearest car within an acceptable walking distance. If successful, the user proceeds to pick up the car; if no car is available, the user may retry after some time.
- **Pickup Event**: A user walks to the reserved car's location and begins using it. This marks the start of a trip and records the walking time from reservation to pickup.
- **Dropoff Event**: A user completes their trip and leaves the car at the destination. The system updates the car's location and battery charge based on the distance traveled. If the battery is low, the car is marked for relocation to a charging station.
- **Relocate Car Event**: A car with low battery is assigned to a relocator (a worker who drives cars to charging stations). The system finds an available relocator and the nearest charging station, then schedules the car's arrival at the station.
- **Arrive at Station with Relocator Event**: A relocator delivers a car to a charging station. The car enters the charging queue or begins charging immediately if a spot is available.
- **Charging Complete Event**: A car finishes charging and its battery is fully restored. The car becomes available again for user reservations.

These events form a continuous cycle: users subscribe, make reservations, use cars, and return them, while the system manages car relocation and charging to maintain service availability.

2 Data Structures

Important Note

For a matter of simplicity, the service area is modeled as a flat 100 km × 100 km square without geographical obstacles. Realistic traffic patterns are simulated through predefined traffic zones that affect vehicle speeds and travel times. The map structure and the relative graph representations and traffic condition can be considered parameters of the simulation model.

2.1 Service Area and Traffic Zones

The car-sharing service operates within a certain urban area divided into multiple traffic zones. Each zone has distinct traffic characteristics that affect vehicle travel times and user experience. An example of a map used for most of our simulations is the following:

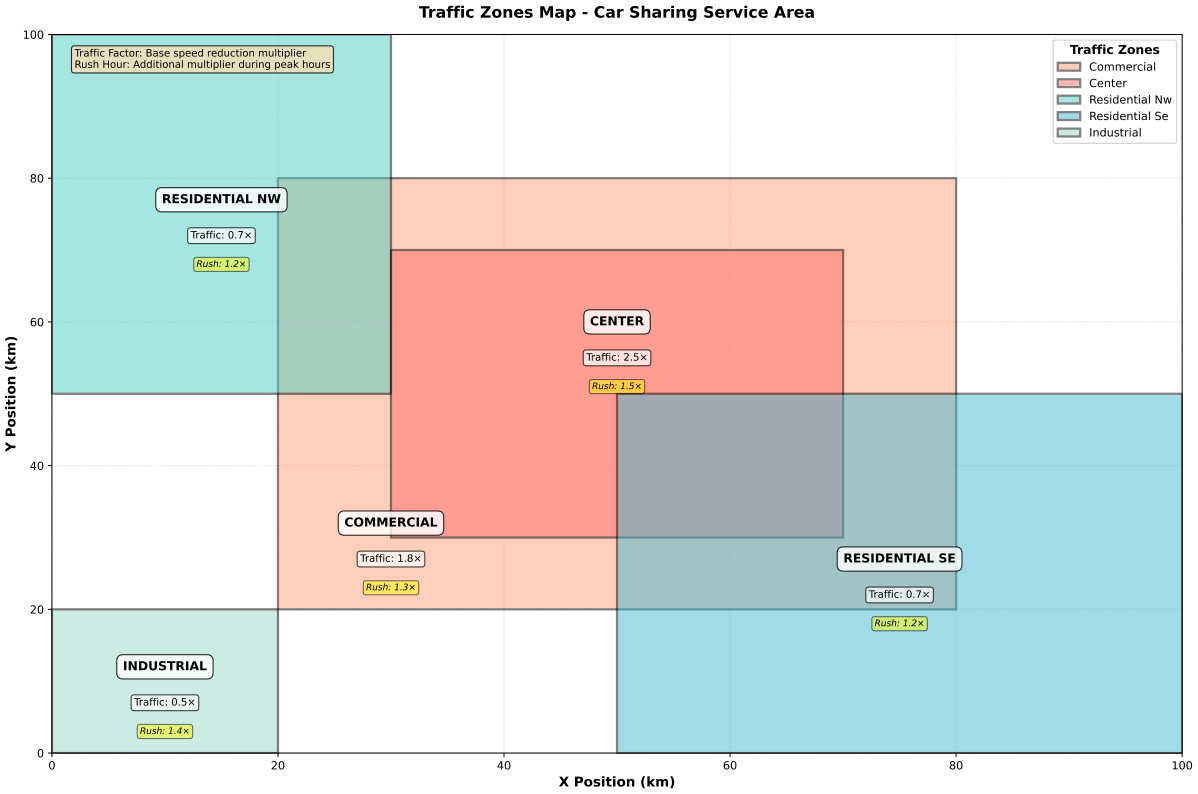

Figure 1: **Traffic zones in the service area** - The map shows five distinct zones with different traffic patterns. The traffic factor, when present, indicates how much slower vehicles move compared to free-flow conditions, while the rush hour multiplier shows additional slowdown during peak hours.

These traffic patterns are incorporated into the road network simulation to provide realistic travel time estimates throughout the day.

2.2 Road Network Graph

To calculate realistic distances and travel times, the simulation uses a **graph structure** to represent the city's road network. This graph consists of:

- **Nodes** (intersections): Points where roads meet, placed on a grid across the city
- **Edges** (roads): Connections between nodes that vehicles can travel along

Important Note

While, in this simulation, users can park anywhere in the service area, it is assumed that the time taken will be computed as the time needed to travel along the edges from the nearest node to the user's location to the nearest node to the destination location. This allows us to simulate realistic travel times based on actual road layouts and traffic conditions, rather than straight-line distances.

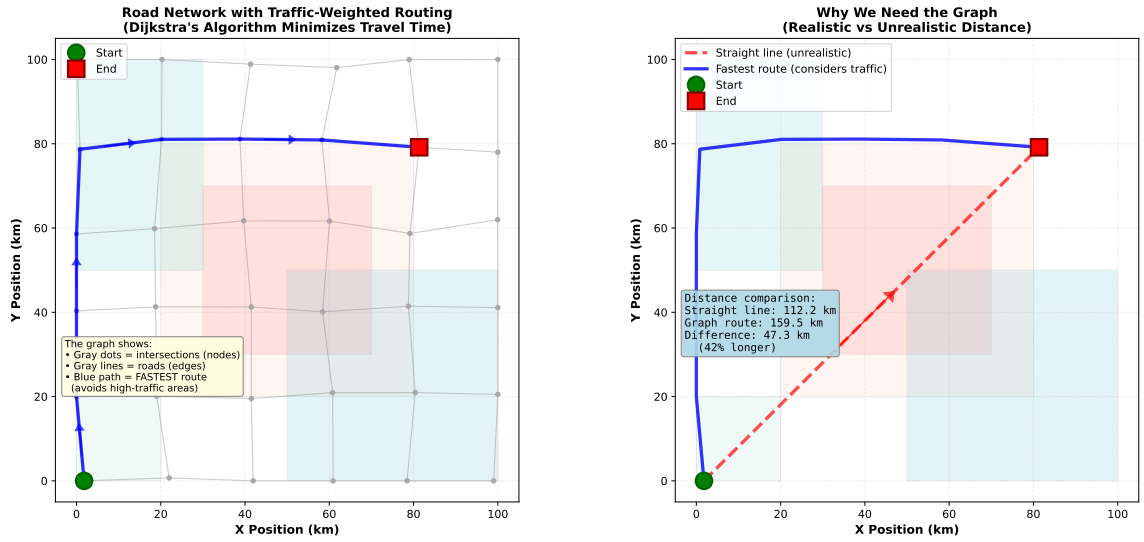

Figure 2: **Road network graph and route calculation** - Left: The graph structure with nodes (intersections) and edges (roads). The blue path shows the route calculated by Dijkstra's algorithm. Right: Comparison between unrealistic straight-line distance and realistic graph-based distance following actual roads.

1. When a user needs to travel from point A to point B, the system finds the nearest graph node to each point
2. Each edge (road) is weighted by **travel time**.
3. Dijkstra's algorithm finds the path with the **minimum total travel time**
4. Roads in high-traffic zones get higher time weights, so the algorithm prefers faster alternative routes

This approach means cars automatically avoid congested areas when faster routes exist, just like real drivers with GPS navigation.

3 Key Parameters

The simulation of such a complex system has a huge variety of parameters, however in this experimentation, we will focus on this subset of key parameters:

- **Fleet Size**: amount of cars available in the system
- **Maximum Pickup Radius**: maximum distance a user is willing to walk to reach a car
- **Maximum amount of users**: limit on the total number of users in the system
- **User Subscription Rate**: rate at which new users join the platform, if set to a very high value, all users sign in at the beginning of the simulation
- **User Reservation Rate**: rate at which each user makes car rental requests
- **Number of charging stations**: total number of charging stations available in the service area
- **Number of relocators**: total number of relocators responsible for moving cars between locations

4 Performance Indicators

To evaluate the performance of the car-sharing system, we will track the following key performance indicators (KPIs):

- **Fleet Utilization**: the percentage of the fleet in use at any given time, a car sharing company may want it to be both high enough to ensure profitability but not too high to avoid user dissatisfaction due to unavailability
- **Successful Reservations Rate**: the ratio of successful car reservations to total reservation attempts, it should be as high as possible
- **Average Walking Distance**: the average distance users walk to reach their reserved cars
- **Average Attempts per Reservation**: the average number of attempts users make before successfully finding an available car, should be as close as possible to 1

5 Implementation

Lab 2 Assignment:

The goal of this lab is to implement in Python the simulator of the model developed in LAB 2. You should also test your simulator, in a few scenarios and assess the impact of some input parameters (the choice of the subset of parameters to test is left to you) on the dynamics.

As shown in the course this discrete-event simulation is implemented using a **Future Event Set (FES)**, which determines when and how things happen in the simulation.

5.1 The Future Event Set (FES)

The FES is a **priority queue** that stores all scheduled events sorted by time.

How the FES works:

1. Events are added to the FES with a specific time when they should occur
2. The FES automatically keeps events sorted by time (earliest first)
3. The simulator repeatedly takes the next event from the FES and processes it
4. Processing an event may create new events, which are added back to the FES

Example: When a user subscribes at time 100, the simulator schedules a "reservation event" at time 120. When that reservation succeeds, it schedules a "pickup event" at time 125, which then schedules a "dropoff event" at time 180. If the dropoff causes the car to need charging, a "relocate car event" is scheduled at time 185, and so on.

5.2 Main Simulation Loop

The simulator follows a simple pattern:

1. **Initialize**: Create all entities (cars, stations, relocators, road network)
2. **Schedule initial events**: Add first user arrival to FES
3. **Main loop**: Repeat until simulation end time:
    - Get the next event from FES (the one with earliest time)
    - Check if event time exceeds simulation end time - if yes, stop
    - Execute the event (call its associated function)
    - The event function updates system state and schedules new events
4. **Finish**: Collect and report final metrics

5.3 Project Structure

The python code is organized into logical modules that separate different concerns, following best practices of the object oriented programming paradigm:

- **Entities/**: Classes representing active participants in the system (class definitions only)
    - `Car.py` - `Car` class: manages vehicle location, battery level, and status, implementing the corresponding entity.
    - `user.py` - `User` class: handles user initialization and attributes, implementing the corresponding entity.

  - `car_relocator.py` - `CarRelocator` class: manages relocator availability and task assignment, implementing the correspoinding entity.

- **infrastructure/**: Classes for physical infrastructure (class definitions only)
  - `road_map.py` - functions to build the road map graph and calculate routes and travel time
  - `charging_station.py` - handles all interactions related to charging stations

- **Core simulation files**:
  - `simulator.py` - Runs the simulation, implements the FES loop and coordinates all events
  - `events.py` - Here all the event functions are defined.
  - `config.py` - Configuration parameters and constants supports the loading of parameters from YAML files.
  - `metrics.py` - `Metrics` utilities to collects statistics and performs transient analysis

5.4 Running instructions

6 Experimental Scenarios and Results

To evaluate the car-sharing system under different operating conditions, we designed several experimental scenarios focusing on the impact of fleet size and user demand. All experiments were conducted with a 2-week simulation period (20,160 minutes) to allow the system to reach steady-state operation and provide statistically significant results.

6.1 Experimental Design

6.1.1 User Arrival Pattern

All scenarios use an **instant arrival pattern** where all users subscribe to the service at the simulation start. This approach:

- Simulates a mature car-sharing service with an established user base
- Eliminates the transient phase of gradual user adoption
- Provides immediate system stress testing
- Allows direct comparison of steady-state performance across scenarios

After arrival, users make reservation requests at a rate of 0.02 requests per hour (approximately one request every 50 hours per user), representing realistic usage patterns for occasional car-sharing service users.

6.1.2 Test Scenarios

Four scenarios were designed to systematically evaluate the system:

Scenario	Users	Cars	Ratio
Baseline	20	20	1:1
500 Users, 50 Cars	500	50	10:1
2000 Users, 50 Cars	2000	50	40:1
5000 Users, 200 Cars	5000	200	25:1

Table 1: Experimental scenarios testing different user-to-car ratios and system scales.

All scenarios maintain consistent infrastructure:

- 5 charging stations distributed across the service area
- 3 relocators for car redistribution
- Maximum pickup distance: 3 km

6.2 Performance Results

6.2.1 Comprehensive Metrics Table

The following table presents all key performance indicators across the four experimental scenarios:

Metric	Baseline	500u/50c	2000u/50c	5000u/200c
Service Quality Indicators				
Success Rate	83.0%	91.5%	80.8%	81.2%
Total Reservations	459	3,314	13,190	13,054
Avg Attempts	1.00	1.01	1.05	1.05
Total Trips	379	3,029	10,643	10,578
Fleet Performance Indicators				
In-Use Rate	0.1%	11.5%	41.4%	42.3%
Charging Rate	0.1%	5.1%	19.6%	19.3%
Idle Rate	99.8%	83.4%	39.0%	38.4%
Operational Indicators				
Avg Trip Distance	6.3 km	6.0 km	6.1 km	6.0 km
Total Distance	2,390.9 km	18,248.4 km	64,436.1 km	63,555.9 km
Charging Sessions	45	398	1,455	1,433
Avg Queue Length	0.50	1.05	2.36	2.27

Table 2: Complete performance metrics for all experimental scenarios.

6.2.2 Key Findings

Based on the data in Table 2, we can describe the scenarios based on user and car (c) counts.

- The **Baseline** scenario shows a system with minimal activity, characterized by a 99.8% Idle Rate. The specific user and car counts are not defined but activity is negligible.
- The **500u/50c** scenario introduces 500 users and 50 cars. This configuration results in the highest Success Rate (91.5%) but a relatively low fleet utilization (11.5% In-Use Rate).
- The **2000u/50c** scenario increases the user load to 2000 users while keeping the fleet at 50 cars. This high user-to-car ratio leads to a significant increase in fleet utilization (41.4%) but also a drop in the Success Rate to 80.8%.
- The **5000u/200c** scenario scales up both users (5000) and cars (200). Interestingly, despite the four-fold increase in cars compared to the previous scenario, key metrics like the Success Rate (81.2%) and In-Use Rate (42.3%) remain remarkably similar to the 2000u/50c case.

7 Conclusion

The experimental results presented in this report illustrate the complex dynamics between fleet size, user demand, and service performance. The data indicates that increasing the scale of the system, as seen when transitioning from the 2000u/50c to the 5000u/200c scenario, does not necessarily lead to proportional or intuitive changes in key performance indicators.

The similarity in performance between the last two scenarios, despite significantly different resource allocations, suggests that system behavior is governed by non-linear factors. These preliminary findings raise questions about operational bottlenecks and resource efficiency. Further investigation is required to fully understand the observed dynamics and to determine effective strategies for balancing service quality with fleet utilization.