Ride-Sharing Simulation for Optimisation

Bernhard J. Berger
Hamburg University of Technology
Hamburg, Germany
bernhard.berger@tuhh.de

Christina Plump*
University of Bremen
Bremen, Germany
cplump@uni-bremen.de

Rolf Drechsler†
University of Bremen
Bremen, Germany
drechsler@uni-bremen.de

ABSTRACT

Optimisation research requires possibilities to test and evaluate new algorithms and algorithm adaptions. Two interesting problem classes—relevant to society—that covers shortest path problems and resource allocation in a very dynamic environment, with the possibility of running endlessly, are ride-hailing and ride-sharing. This paper presents an extensible simulation framework for ride-hailing and ride-sharing problems that allows researchers to create solutions in any program language, provides a graphical user-interface, and automatic evaluation.

CCS CONCEPTS

• **Applied computing** → **Transportation**; • **Theory of computation** → **Continuous optimization**; • **Computing methodologies** → *Discrete-event simulation.*

ACM Reference Format:
Bernhard J. Berger, Christina Plump, and Rolf Drechsler. 2026. Ride-Sharing Simulation for Optimisation. In *Genetic and Evolutionary Computation Conference (GECCO Companion '26), July 13–17, 2026, San Jose, Costa Rica.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In today's fast-paced world, optimisation is used for addressing real-world logistical challenges such as resource allocation and scheduling. While static optimisation techniques have been extensively studied, the dynamic nature of real-world environments demands online and dynamic algorithms capable of adapting to changing conditions while providing near-optimal solutions. On-demand ride-hailing and ride-sharing services exemplify these online, real-time optimisation challenges. Platforms like Uber and Lyft have transformed transportation by connecting drivers with passengers via mobile apps, offering convenience, affordability, and sustainability through ride-sharing. Research has shown that ideas such as ride-sharing have a positive effect on urban transportation emissions [7] and is an active research topic [1–3, 5, 6]. These services must manage dynamically arising customer requests, ensuring quick response times while balancing customer satisfaction, operational efficiency, and sustainability.

Essentially, the problems mentioned consist of several optimisation problems. First, there is an underlying road network on which optimisation algorithms must solve the problem of finding the shortest route using a cost function adapted to the given optimality criteria. Second, solution algorithms must also solve the allocation problem of existing vehicles to customer requests. The aim here is to minimise waiting times and to assess whether requests can be fulfilled at all under the given circumstances, as customers may also have deadlines by which they wish to be served. The dynamic nature of the problem adds further challenges. Customer requests are not known in advance, leading to an ongoing—and if wanted even endless—optimisation process. Solutions found may cease to be optimal due to new events; for example, the optimal route may change due to traffic jams, and assignments may become suboptimal due to new taxis or customer requests.

While such problems are highly interesting for research, it is not simple to find problem instances that can be used for developing and testing optimisation solutions. We developed a simulation framework for ride-hailing and ride-sharing problems within a student project on optimisation. The current implementation offers a dynamic multi-objective optimisation problem, where optimisers need to balance costs, CO_2 emissions, and customer satisfaction (measured by driving and waiting time metrics). In addition, the optimisation algorithm should generate explanations that clarify their decisions to different stakeholders (taxi agency, taxi driver, and customer). The framework is used as a basis for the GECCO'26 *Explainable ride-sharing optimisation for sustainable traffic organisation* competition [1] and is available open-source[2].

The framework offers several advantages: (1) clear separation of simulation and optimisation, (2) open architecture to allow extensibility, (3) graphical user-interface for interactive usage and evaluation, (4) automatic evaluation for comparing different solutions, and (5) dynamic generation of problem instances that leads to a less likely overfitting of the developed optimisation solution to a specific problem instance.

The remainder of the paper is structured as follows: Section 2 introduces the ride-hailing and ride-sharing simulation framework. Next, Section 3 describes how custom optimisers can be connected to the simulation. Afterwards, Section 4 discusses the current implementation, shortcomings and future improvements. Finally, Section 5 concludes the paper.

2 FRAMEWORK

This section describes the developed simulation framework and describes extension possibilities. First, Section 2.1 describes use-cases of the simulation framework. Section 2.2 describes the general system architecture and how it supports the use-cases. Finally, Section 2.3 describes the current implementation of the simulation.

*Also with DFKI GmbH—Cyber-Physical Systems.
†Also with DFKI GmbH—Cyber-Physical Systems.

GECCO Companion '26, July 13–17, 2026, San Jose, Costa Rica

ACM ISBN 979-8-4007-2488-6/2026/07
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] See https://gecco-2026.sigevo.org/Competition?itemId=8269
[2] Please visit https://gitlab.informatik.uni-bremen.de/evoal/vehicle-routing-problem/framework

2.1 Use Cases

We identified three main use-cases for the framework:

Optimiser Implementation The main goal of the framework is to allow researchers to implement new algorithms in different programming languages that can use the very same simulation to make the comparison as fair as possible. This requirement results in the need of a strict separation of the simulation framework and potential optimisers, while, at the same time, it is necessary that a portable and easy-to-use interface exists. Additionally, it is helpful for algorithm development to visually see the impact of the optimiser in terms of domain changes and evaluation criteria (metrics). (Re-)running (pre-defined) existing scenarios supports researchers to see the impact of algorithmic changes by repeating a scenario, while running newly generated scenarios allow avoiding overfitting to a limited set of scenarios.

Simulation Extension Simulation extension is a common use-case to allow researchers to investigate different aspects of the problem domain, for example, by extending the problem to support proper traffic jams. Additionally, researchers may want to extend the simulation by a corresponding simulation (or connecting to a corresponding existing one). The framework requires an open architecture that allows new internal or external components to interact with the existing simulation to support this use-case.

Evaluation and Comparison The use-case of evaluation and comparing algorithms poses different requirements. First, the simulation should run deterministicly and automatically, using a given scenario. The simulation has to collect evaluation metrics to make different optimisers comparable. In this use-case, a graphical user-interface is not necessary, but it is helpful if the execution speed of the simulation can be sped up, to execute long-time simulations that span several days in a shorter period of time.

2.2 Architecture

Based on the requirements listed above, the simulation is split into three processes. Figure 1 gives a deployment overview. First, we have a Java-based simulation that reads three JSON files for configuration. The network files contain a part of the open streetmap data[3], which is a property graph that contains intersections (nodes) and roads (edges), as well as the corresponding metadata, e.g., the length of a road or the speed limit. The optional scenario files contain scenario-specific information, e.g., existing vehicles and customers, as well as corresponding parameters, e.g., the size of a car.

The simulation process is responsible for simulating the world, e.g., moving vehicles and customers. Furthermore, it provides an interface for optimisers, the user-interface, and the evaluation process to interact with the simulation. The central component is a discrete timer that emits tick events for every simulated second. The agents that are simulated use these time ticks to update their simulation state and publish these changes as corresponding events that other components can consume, e.g, the vehicle manager. This component holds all vehicles and moves them based on the time passed. Additionally, it calculates move-related metrics, such as the total distance travelled and the CO_2 emissions. The event-based

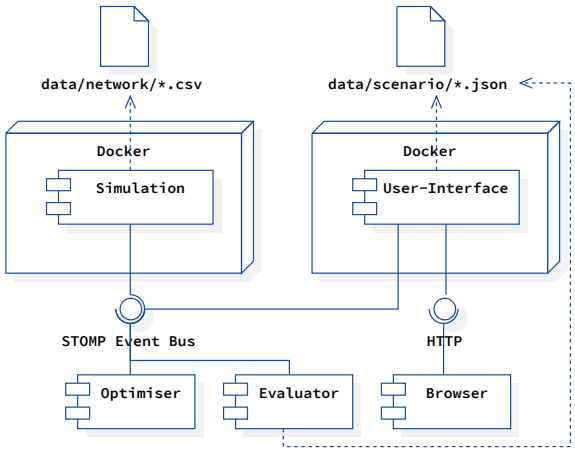

Figure 1: Deployment Overview

architecture allows an easy extension of the simulation, since external components can subscribe to simulation events, e.g., when a new customer request arrives, or when a vehicle moves. Additionally, external components can publish events to the simulation, e.g., to assign a customer to a vehicle. The event bus is exposed using the STOMP protocol[4], allowing any client to connect to the simulation. The optimiser, our user-interface, and the evaluation process connect to this event bus[5].

The optimiser can be implemented in any programming language, as long as it can connect to the event bus and send and receive messages in the expected format. The optimiser receives information about the current world state, e.g., the current position of vehicles and customers, and can reply by sending route planning events.

The user-interface is an Angular-based web application that can be accessed via a web browser. It gives a visual representation of the current world state, e.g., by showing an OpenStreetMap map, and the current position of vehicles and customers. Additionally, it allows users to interact with the simulation, e.g., by adding new requests or vehicles. It has a metrics dashboard to show the current evaluation metrics, and a scenario manager to load and run different scenarios. Figure 2 shows a screenshot of the user-interface. The screenshot shows an excerpt of the metrics overview on the left-hand side. In the middle of the screenshot, the map is shown, including a taxi, a customer who awaits collection and the corresponding destination, and the planned taxi route. On the right-hand side, the screenshot shows a list of all current and past requests. It also shows status information on the request, such as, how many customers are related to the request, the number of customers waiting for a taxi, the number of customers currently in taxis, the number of customers who reached their destination, and how many customers were not served.

The evaluation process steers the simulation execution entirely automatically using the same means then the user interface. After connecting to the event bus, it first uploads the road network to use

[3]Compare https://www.openstreetmap.org

[4]Compare https://stomp.github.io/

[5]The simulator filters events to ensure certain events are sent only by simulation components, or the user-interface

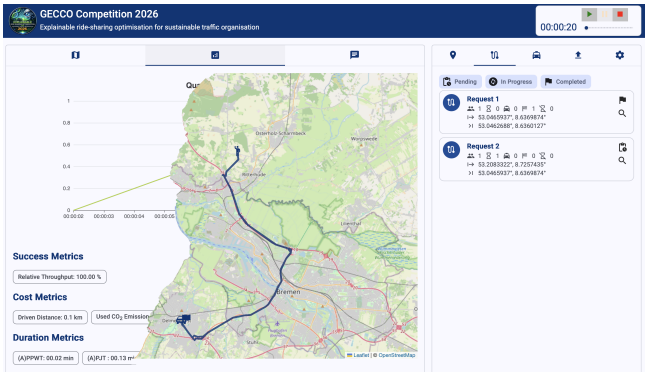

Figure 2: A screenshot collage of the graphical user-interface

and then transmits a pre-recorded scenario[6] to the simulation. The execution speed is set to a configurable value, e.g., the simulation of 60 simulation seconds in 1 real-world second, allowing to test the performance of solutions. After the initialisation phase, the evaluation process protocols the events, including the calculated performance metrics. Finally, when the simulation simulated a pre-defined time span, e.g., 24 hours, the evaluation process stops the simulation.

We use Docker[7] to containerise the simulation and user-interface to make it easy to set up and run the framework. The optimiser and evaluation process can be run locally, or also be containerised.

2.3 Simulation

The simulation is implemented in Java and uses the Spring Boot[8] framework. The simulation is event-based, which means that the components are loosely coupled and interact via a message bus. The simulation consists of six core simulation components. The most fundamental component is the *TimeManager*. It holds the simulation state—whether it is stopped, initializing, running, or paused—and sends *tick* events for every simulated second. The *WorldManager* loads and manages the underlying road network, and sends information on property changes within the road network, e.g., a changed speed limit or changes in the network structure. The *PersonManager* deals with everything that is related to persons in the simulation. Currently these aspects are keeping track of the persons position, and calculating waiting and driving times. The *VehicleManager* controls and simulates all cars. It simulates every movement of vehicles, the loading state, and getting on and off a vehicle. The *RequestManager* keeps track of all requests and checks if they are fulfilled and calculates respective quality metrics. Finally, the *ScenarioManager* keeps track of pre-defined scenarios and sends scenario events when necessary.

Using JSON as the event serialisation format allows clients written in any programming language to consume and process these events. Listings 1 shows the event that is sent by the simulation when a taxi was added to the taxi fleet. This event is created by the *TaxiManager* after checking the correctness of an add taxi event. The event states the taxi id, which is used in subsequent taxi-related events, such as route planning events (sent by the optimiser), or movement events (sent by the simulation). The intersection id relates to a network node (intersection) where the taxi is currently located at. The properties contain additional—mostly simulation related information—of the taxi, such as the passenger capacity, the maximum vehicle speed, or CO_2 consumption information.

```
{
  "category"        : "taxi-fleet",
  "name"            : "added-taxi",
  "id"              : "taxi-1",
  "intersection-id" : 243863178,
  "properties" : {
    "maximum-capacity"           : 1,
    "maximum-speed"              : 100,
    "energy-efficiency-constant" : 0.87,
    "co2-factor"                 : 310.0
  }
}
```

Listing 1: Simulation event for informing clients that a new taxi (vehicle) was added

2.4 System Requirements

For running the simulation, a system that supports Docker is required. One aspect that influences the system requirements is the simulation size (number of vehicles, requests, and so on) as the simulation has to simulate the movement of all existing vehicles. Internally, the simulation checks, if the simulation clock can keep up with the real-time clock (if all simulation steps can be executed within one second). If this is not the case, the simulation logs this information. The second influencing factor is the deployment setup. For calculation intensive tasks it is necessary to run the simulation and the optimisation on different machines. In general, a standard computer is sufficient for running a simulation. A MacBook Pro M5 with 32GB of memory is able to run a simulation spanning 24 hours (simulating 60 seconds in one real-world second) with roughly 2000 requests and 100 cars parallel to the optimisation.

3 OPTIMISER IMPLEMENTATION

In this section, we describe how optimisation researchers can integrate their own optimisation algorithms. This description is based on the tutorial available on the competition page [9] and implements a greedy algorithm that handles requests by assigning them to the next available vehicle.

The tutorial optimiser is also implemented in Java (but could be implemented in any language). The connection to the simulation is established using an existing Java library that supports the STOMP protocol. Listing 2 shows two functions that first establish the connection to the simulation and then register for simulation events. The simulator requires clients to set their role in the headers for event filtering purposes.

[6] The repository contains a scenario generator that generates scenarios based on a configuration. To this end, it uses information from open streetmap and generates corresponding events.
[7] See https://www.docker.com/
[8] See https://spring.io/projects/spring-boot
[9] See https://evoal.de/docs/competition26/tutorial

```java
public void connect() {
  final StompHeaders headers = new StompHeaders();
  headers.add("role", "OPTIMIZER");
  stompClient.connectAsync(
    "ws://localhost:8088/simulation-websocket",
    new WebSocketHttpHeaders(), headers, this);
}

@Override
public synchronized void afterConnected(
    StompSession session,
    StompHeaders connectedHeaders) {
    session.subscribe("/topic/simulation-events", ...);
}
```

Listing 2: Establishing the Simulation Connection

The second parameter of the *subscribe* call expects an event receiver. In our reference implementation, we have a custom event dispatcher that consumes all events and sends them to registered consumers. Components can register for events based on the event category and name (we specify these combinations as *"category:name"* in the following). The optimiser has, for instance, a world state component that keeps track of a) the road network, b) vehicles, and c) customers. It registers for *"vehicle:passed-intersection"* events to update the position of vehicles within the road network. The schedule planning component registers for *"request:ride-request-received"* events, plans the optimal solution based on the world state, and responds with a *"taxi-fleet:plan-route"* event that contains the detailed route of a vehicle to serve the request.

4 DISCUSSION

This section discusses the proposed framework and relates it to current research.

The proposed framework offers a wide range of advantages. Besides providing a timely and interesting optimisation problem, it allows a fair comparison of different optimisation algorithms to a well-defined problem. The used event-based architecture allows extending the simulation and implementing optimisation algorithms in any programming language and technology. Due to its open architecture, the simulation is not only bound to ride-hailing and ride-sharing problems but can be extended to similar problem classes. To this end, it is possible to add additional simulation components using the event bus, e.g., a traffic jam simulation. Such a simulation could monitor the vehicle moves and calculate throughput metrics for the roads. Based on these metrics it can publish events that reduce the maximum speed for heavily used roads. Currently, we are working on an adaptation that includes container scheduling problems. The framework has, of course, also disadvantages. The technical setup is more complicated in comparison to problems, where instances can be specified using static CSV files. Unfortunately, this is always the case when the problem is dynamic and not known in advance. In such cases, it is necessary to keep track of the problem state. The simulation setup is simplified as much as possible by providing pre-built Docker images in our GitLab container registry [10] and a ready-to-use Docker compose configuration to start the simulation and the user-interface.

There are alternatives to our presented approach. One such simulation for ride-hailing and ride-sharing problems is `RidePy`[11] is an open-source simulation that is implemented using C++ and Python [4]. In contrast to the described framework, the simulation does not provide a graphical user interface, supports only Python-based optimisation, and is less extensive regarding evaluation criteria. Another existing simulation is the *Ride Sharing Simulator*[12]. This framework is a Python-based simulation as well that follows a monolithic architecture approach. All parts (the simulation, the optimisation, the evaluation) are running within the same process.

5 CONCLUSIONS

This paper, descibed a Java-based ride-hailing and ride-sharing simulation framework for testing and evaluating optimisation algorithm ideas. The simulation offers a web technology-based, event-based interface for optimisers and offers a timely, interesting, and challenging problem. Furthermore, the simulation is extensible using the provided event bus, allowing to add additional challenges to the optimisation problem. Additionally, the paper gives a short overview of how to connect optimisers to the simulator, which can be implemented in any programming language. The framework is used as the basis of the GECCO'26 *Explainable ride-sharing optimisation for sustainable traffic organisation* competition.

ACKNOWLEDGEMENT

Supported by Deutsche Forschungsgemeinschaft (DFG) GRK 2972 CAUSE – Project Number: 513623283. Additionally, we would like to thank the students of the ElO4Future project at the University of Bremen for their contributions to the simulation framework.

REFERENCES

[1] Negin Alisoltani, Younes Delhoum, Mostafa Ameli, and Mahdi Zargayouna. 2024. Enhancing Urban Mobility Through Peer-to-Peer Ride-Sharing: A System-Wide Impact Assessment. In *Proceedings of the 2nd ACM SIGSPATIAL Workshop on Sustainable Urban Mobility (SIGSPATIAL'24)*. ACM, 18–26.

[2] Mariem Ayari, Sonia Nasri, Hend Bouziri, and Wassila Aggoune-Mtalaa. 2025. A Local Search Deep Q-Network to Optimize On-Demand Tranportation Problems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO'25 Companion)*. ACM, 763–766.

[3] Roman Engelhardt, Florian Dandl, Aledia Bilali, and Klaus Bogenberger. 2019. Quantifying the Benefits of Autonomous On-Demand Ride-Pooling: A Simulation Study for Munich, Germany. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2992–2997.

[4] Felix Jung and Debsankha Manik. 2024. RidePy: A fast and modular framework for simulating ridepooling systems. *Journal of Open Source Software* 9, 97 (May 2024), 6241.

[5] Divyanshu Singh, Ashman Mehra, Kavya Makwana, Snehanshu Saha, and Santonu Sarkar. 2025. Altruistic Ride Sharing: A Framework for Fair and Sustainable Urban Mobility via Peer-to-Peer Incentives.

[6] Shigeo Takahashi, Ryoya Yoshimoto, Yuki Tanioka, and Kazuo Misue. 2025. Interactive Exploration of Approximate Solutions for On-Demand Ride-Sharing Transportation. In *Proceedings of the 18th International Symposium on Visual Information Communication and Interaction (VINCI 2025)*. ACM, 1–5.

[7] Lisa Winkler, Drew Pearce, Jenny Nelson, and Oytun Babacan. 2023. The effect of sustainable mobility transition policies on cumulative urban transport emissions and energy demand. *Nature Communications* 14, 1 (April 2023).

[10] See https://gitlab.informatik.uni-bremen.de/evoal/vehicle-routing-problem/gecco26-competition

[11] Available at https://github.com/PhysicsOfMobility/ridepy.

[12] Available at https://github.com/HKU-Smart-Mobility-Lab/Ride-sharing-Simulator