UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID
Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Universitat Politècnica de València
Departamento de Sistemas Informáticos y Computación

Improving Urban Mobility with Game Theory Techniques

MASTER'S DEGREE FINAL WORK

Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

Author: Martí Gimeno, Pasqual

Tutor: Julián Inglada, Vicente Javier
Jordán Prunera, Jaume Magí

Course 2019-2020

Resum

La millora del trànsit urbà cobra cada vegada més rellevància a mesura que la població de les ciutats augmenta. Al nostre treball fem ús de SimFleet, un simulador multiagent de flotes urbanes, per a estudiar l'optimització del trànsit urbà. Per a trobar solucions pràctiques a aquest problema, utilitzem agents racionals autointeressats que aconsegueixen simulacions més realistes. Per a la coordinació dels agents, dissenyem i implementem un mètode basat en teoria de jocs que obté un equilibri; una solució de la qual cap agent està incentivat a desviar-se. A més, ampliem les funcionalitats de SimFleet implementant diversos mòduls que faciliten la creació d'escenaris de simulació complexos i introdueixen un nou tipus de flota urbana, fent-lo així un simulador més complet.

Paraules clau: Mobilitat urbana, Teoria de Jocs, Sistema multiagent, Planificació, Simulació, SimFleet

Resumen

La mejora del tráfico urbano cobra cada vez más relevancia a medida que la población de las ciudades aumenta. En nuestro trabajo hacemos uso de SimFleet, un simulador multiagente de flotas urbanas, para estudiar la optimización del tráfico urbano. Para encontrar soluciones prácticas a dicho problema, utilizamos agentes racionales autointeresados que consiguen simulaciones más realistas. Para la coordinación de los agentes, diseñamos e implementamos un método basado en teoría de juegos que obtiene un equilibrio; una solución de la que ningún agente está incentivado a desviarse. Además, ampliamos las funcionalidades de SimFleet implementando diversos módulos que facilitan la creación de escenarios de simulación complejos e introducen un nuevo tipo de flota urbana, haciéndolo así un simulador más completo.

Palabras clave: Movilidad urbana, Teoría de Juegos, Sistema multiagente, Planificación, Simulación, SimFleet

Abstract

The improvement of urban traffic is increasingly relevant as the population in cities increases. In our work we use SimFleet, a multi-agent urban fleet simulator, to study the optimization of urban traffic. Aiming to find practical solutions to this problem, we introduce rational self-interested agents which achieve more realistic simulations. For the coordination of the agents, we design and implement a method based on game theory that obtains an equilibrium; a solution from which no agent is incentivized to deviate. Besides, we broaden SimFleet's functionalities by implementing several modules which ease the creation of complex simulation scenarios and introduce a new type of urban fleet, making it a more complete simulator.

Key words: Urban mobility, Game Theory, Multiagent system, Planning, Simulation, SimFleet

Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

With more than half of the world's population living in cities, the list of challenges for keeping them sustainable has grown. "A smart sustainable city is an innovative city that uses ICTs (Information and Communication Technologies) to improve quality of life, the efficiency of urban operations and services and competitiveness while ensuring that it meets the needs of present and future generations concerning economic, social, environmental and cultural aspects"[1].

The population in cities has diversified and people from all kinds of backgrounds may have to live in a city if they want to have certain services that today we consider essential, such as the internet access. This has awakened new concerns among citizens and, consequently, city councils. On the one hand, they want to reduce air pollution by promoting the use of bicycles, public transport and even electric vehicles instead of the conventional gasoline-powered car. On the other hand, the existence of green areas throughout the city is valued; areas that beautify the appearance of the city and are related to a better quality of life. All of this seems to have influenced urban traffic, making it evolve to focus on the people rather than the vehicles. For instance, many municipalities are restricting the traffic inside their town's center, increasing the space pedestrians have to walk as well as the air quality.

The urban traffic system, which was already complex, is becoming more entangled as citizens needs and concerns evolve. With the popularization of cab companies and take-out applications, we have nowadays more chauffeur-driven rental and delivery vehicles constantly traveling around cities than ever before. All these traffic interactions are not trivially sorted out which causes experts to be constantly researching for new ways of optimizing the traffic flow in urban settlements. However, this increase in complexity has been accompanied by a technological revolution, the so-called "smart" devices. Today, almost every electronic device has its smart version, which connects it to the Internet and uses all the data it provides to improve its user experience. Similarly, cities are evolving into smart cities that control parameters such as traffic status, the influx of people in different areas or on public transport and even the quality of their air, in real time.

[1]This definition was provided by the International Telecommunication Union (ITU) and United Nations Economic Commission for Europe (UNECE) in 2015

Therefore, as more city services become intelligent, we have more and more data that we can use both to better characterize the urban traffic problems and to advance in their resolution. Through the use of artificial intelligence, we can give more potential to such data by using it in a more problem-oriented way. In line with this, what areas of artificial intelligence would be most useful for working on urban mobility? How can we make efficient use of data to arrive at solutions? Is it possible to reproduce systems as complex as cities virtually to test improvements on them? Our work explores these research questions with the intention of finding practical improvements to the challenges of urban mobility.

1.1 Motivation

This work is motivated by SimFleet and its high potential as an urban traffic simulator.

The urban traffic problem becomes more complex as the population in cities increases and the needs of citizens evolve. However, we have access to more data trough the intelligent city services which may aid in the improvement of urban mobility if used adequately.

While the handling of such data can lead to solutions, the changes these imply cannot be applied without considering the impact they may have on the city's inhabitants. In addition, the variety of scenarios urban traffic offers is massive, since the urban domain counts with many different users. From fleets of taxis to privately owned vehicles, all elements must be taken into account when researching urban traffic optimization. Through the use of simulators we manage to reproduce a small part of the real world in a virtual way, which allows us to modify it as we wish. We can test all kinds of changes or improvements without the need to implement them or the risk of causing negative effects on people's lives. This offers a perfect working area for exploring solutions to the problem of urban traffic, as these are often expensive and costly to implement.

For an accurate simulation, however, the interactions between the elements of the system to replicate must be accurately reproduced. In the case of simple systems, this can be achieved by the simulator alone. In contrast, for more complex scenarios (such as the urban traffic one) it is better to decentralize the simulation load. For this, a multi-agent system seems to be the obvious choice. Agents are pieces of software that are inspired by human reasoning: capturing signals from their environment and reacting appropriately, communicating with other agents, making their own decisions, etc. This makes the agents more than suitable for modeling the different users of a city. Pedestrians, private vehicles, taxis, public transport, delivery vehicles, etc; all types of users can be represented by an agent that adapts its actions and its way of interacting, both with the environment and with other users, accordingly. In addition, there is a great amount of research on communication, coordination, negotiation and conflict resolution between agents. All these areas can be applied to improve urban traffic.

SimFleet combines the possibilities offered by simulators with the flexibility of multi-agent systems, offering an ideal framework for the development and testing of solutions for improving urban traffic. However, SimFleet has a lot of room for improvement, as it is still in an early stage of development. Motivated by this, we design and implement different modules that improve SimFleet, enriching its functionalities.

While some of the improvements are focused on facilitating the use of the software and broadening its range of systems available for simulation, there is another one that explores making the simulations more realistic by basing them on rational human behavior. For this, we make use of self-interested agents; agents with their own objectives that make their decisions accordingly to complete them. This "selfish" behavior can be a more realistic approach for representing certain types of users of an urban traffic system, such as taxis. Taxis (or other types of chauffeur-driven rental vehicles) are interested in serving the maximum number of customers possible, as they report a certain benefit. These vehicles usually belong to a fleet which, in turn, may belong to a company. However, generally, taxi drivers will give more importance to their own benefit rather than the overall benefit of their fleet, thus adopting a self-interested behavior.

Another example for the use of self-interested agents would be the modeling of electric autonomous vehicles (EAVs). In the smart city of the future, EAVs could provide citizens and visitors with the possibility of making private trips without having to use their own vehicle or contribute to the emission of greenhouse gases, as well as avoiding having to adapt to the public transport's routes. These EAVs would look for the best routes in the city to take their users to their destination, coordinating, in turn, with all the other EAVs to optimize traffic globally.

1.2 Objectives

Our work is motivated by the challenges posed by the problem of urban traffic. In this line, the objectives of the work are focused both on the improvement of SimFleet, the simulation software that we use to explore improvements to urban traffic as well as the research of new methods of coordination of autonomous agents. The coordination of self-interested agents is a relevant topic in the present to make simulations more realistic and, in addition, it has potential to be useful in the near future where autonomous vehicles may be an everyday occurrence.

Following, we present a concrete list of objectives:

- Enhance SimFleet to make it a better simulator.
  - Automatize the creation of simulation configurations.
  - Provide means to create different distribution of agents in a simulation easily.
  - Add new types of urban fleets and adapt the software for their simulation.
- Achieve more realistic simulations.
  - Position of elements of the simulation scenario based on real-world data.

  - Generation of simulation load (agent movement in the simulation) that matches real-world citizen and traffic movement in a city.

- Introduce self-interested agents to SimFleet's simulations.
  - Research the coordination of self-interested agents as a potential future solution to urban traffic.
  - Design and implement a system which solves SimFleet simulation scenarios considering the private interest's of the agents.
    * Allow agents to plan their own execution according to their goals.
    * Coordinate the agents to avoid execution conflicts.
    * Ensure the completion of the simulation.

1.3 Document structure

The document is structured as follows:

Chapter I: Introduction
Introduction of our work, discussion the motivation behind it as well as its objectives.

Chapter II: State of the Art
Description of theoretical concepts and technologies in which our work is based, including some relevant works that research and/or employ them.

Chapter III: Proposal
Overview of our proposal, introducing the different contributions of the thesis and giving them context within the global objectives.

Chapter IV: Load Generators
Introduction of the Load Generators for the automatic creation of simulation configurations for SimFleet.

Chapter V: Free-floating Carsharing System
Introduction of the Free-floating carsharing systems and their implementation in SimFleet which enables it to simulate this type of systems.

Chapter VI: SimFleet Planner
Detailed description of our ad-hoc planner, designed and implemented to return optimal plans which take into account the actions of other agents in the scenario.

Chapter VII: Best-Response Planning Strategy
Presentation of the Best-Response Planning Strategy, a game-theoretic approach to the coordination of self-interested agents in which we base our work. We characterize it both from a general point of view and a concrete one based on our

urban traffic domain.

Chapter VII: Experimentation
Evaluation of the ad-hoc planner and the Best-Response Planning Strategy process against simulation scenarios of various sizes with an analysis of the results in terms of time and memory consumption as well as solution quality.

Chapter IX: Conclusions
Conclusions to this work and presentation of future work with which it could be expanded. We also specify the list of scientific publications associated with this work, the research projects in which this master's thesis is included, and the connection with the taken studies.

Bibliography
Previous works related to the researched area which are referenced by our project.

CHAPTER 2
State of the Art

Before getting into the specifics of the project, we must comment on the different areas of artificial intelligence that are related to and researched by our thesis. In this chapter we briefly describe them, providing the knowledge for a complete understanding of our work.

The chapter is organized as follows: Firstly, we present the preliminaries, with key theoretical concepts in which our work is based. Secondly, we introduce related works which research and/or use the technologies we employed during the development of our thesis.

2.1 Preliminaries

In this section we present some theoretical concepts which are key to have a lower-level understanding of the technologies employed in our work. Particularly, we will describe what an intelligent agent is in the scope of our work, as well as give a basic notion of multi-agent systems and multi-agent based simulations. In addition, we introduce SimFleet, the multi-agent based simulation software which links all our work.

2.1.1. Intelligent Agents and Multi-Agent Systems

During the years, there has been a wide discussion about how to properly define software agents. In contrast with procedures or routines, agents are persistent programs whose execution can vary according to their goals and the characteristics of their environment.

In general, the notion of agent refers to a computer system or software with the following properties, presented in [1]:

- **Autonomy**: Agents function without human intervention, having control over their actions and internal state.
- **Social skills**: Agents have the ability to communicate with other agents or humans through a preset language.

- **Reactivity**: Agents perceive their environment and are able to act accordingly, making decisions in real time.

- **Pro-activeness**: Agents, besides reacting to their environment, take the initiative to perform actions aimed at achieving a concrete goal.

In the artificial intelligence field, the definition of agent is broadened by including a human factor. Agents are seen as computer systems that present the aforementioned properties but are also implemented using concepts which are usually applied to humans. For instance, an agent knowledge system can be described with *beliefs*, *desires* and *intentions*, which are notions that refer to the human mind.

This basic definition of intelligent agents can be extended and/or significantly modified depending on the concrete field of science from which the agent is viewed. For a further discussion in what constitutes an agent, we encourage the reader to check [2].

Having described an agent, we can introduce the concept of *Multi-agent systems*. A multi-agent system is defined by a set of two or more agents that interact with each other. In general, agents will represent a user with different objectives and motivations. For a successful interaction, agents will need to cooperate, coordinate and negotiate among them as if they were human beings.

2.1.2. Multi-Agent Simulation

Simulation refers to the experimentation performed over a system that models a real-world system. Simulations are necessary when the real-world system that wants to be examined is not accessible or simply does not exist yet. On the other hand, the real system might be existent but the experimentation on it non-viable. For instance, if we want to study the effect that different configurations of roads will have on the traffic of a certain city, generally, we will not have the resources to implement every alternative and observe the traffic flow. Also, the impact of that on the life of the citizens would most likely be high, and so the changes should not be implemented until proven useful. Therefore, such problem can be addressed from a simulation perspective, creating a scaled reproduction of the real system, implementing the interactions that take place in it and running the different experiments in this controlled environment. Simulations are appropriate tools to understand systems and make predictions over them.

When trying to reproduce systems with plenty of individuals that interact among them, modeling a *multi-agent simulation* is usual. A multi-agent based simulation model consists of a set of agents that encapsulate the behavior of many individuals. The execution of the model (and therefore all of its agents) replicates the behavior of the global system. This distributed approach is sometimes closer to the real world, since agents can be implemented preserving their self-interest and individuality. As we mentioned above, agents in a multi-agent system need to include coordination and conflict-solving mechanisms in their behaviors which ensure a correct simulation. In [3], authors present a multi-agent simulator for urban segregation, which allows them explore the impacts of different contextual mechanisms on the emergence of segregation patterns.

One of the many software that implement multi-agent based simulations is SimFleet [4], which is focused in the simulation of urban fleets. This software is the essential to our thesis, since most of the work presented in this document is either meant to improve it or inspired by it.

2.1.3. SimFleet

SimFleet [4] is an agent-based urban fleet simulator built on the SPADE platform [5]. This fleet simulator was built to allow complex simulations over cities where a large number of agents interact to perform fleet coordination and management algorithms. SimFleet uses the multi-agent systems paradigm to allow the user to get access to autonomous, pro-active, intelligent, communicative entities. SPADE is a multi-agent systems platform that provides these features using a simple but powerful interface, which is why it was chosen for the development of SimFleet.

SPADE allows the creation of autonomous agents that communicate using the open XMPP instant messaging protocol [6]. This protocol is the standard protocol of the IETF and W3C for instant messaging communication and it (or some variant of it) is used in such important communication platforms as WhatsApp, Facebook or Google Talk. SPADE agents have also a web-based interface to create custom app frontends for agents, which is also used by SimFleet to show how every agent is moving through the city in a map that represents all routes made by agents. For the movement of such agents, SimFleet makes use of OSRM[1], a routing engine for finding shortest paths in road networks. Querying an OSRM server specifying the service *route* and passing origin and destination points returns the shortest route between those two points.

Finally, SimFleet is based on the Strategy design pattern, which allows the user to introduce new behaviors to the SimFleet agents without the need of modifying the simulator. This design pattern is used to introduce new algorithms that follow a common interface. In this case, introducing new coordination algorithms to an agent is as simple as building a *StrategyBehaviour* and loading it at SimFleet startup.

SimFleet also provides some common agent classes that can be used (or inherit from them) to create a simulation. These agents represent the entities involved in a fleet simulator: fleet managers, transports, customers, and service directory. Next, we shortly describe these agent classes and how they interact during the simulation.

Fleet manager agents. SimFleet simulates an environment where there can be different kinds of fleets that provide services in a city. Each fleet has a fleet manager who takes care of the fleet, allows transports to be accepted in the fleet and puts customers and carriers in touch with each other to provide a service. An example of a fleet manager is a taxi company call center or a goods transport operator with trucks.

Transport agents. These agents represent the vehicles that are moving through the city providing services. SimFleet supports any kind of city transport such as cars, trucks, taxis, electric cars, skateboards or even drones. However, the user

[1]http://project-osrm.org/

can customize the kind of transport for its simulations. Transport may or may not belong to a fleet, but belonging to a fleet brings them some benefits like being found more easily and having a coordinator for its operations. Transport agents receive transport requests from customers and, if free, they will pick the customer up (or the package) and drive to its destination. However, before attending a request, a transport agent will make sure it has enough autonomy to do the whole trip. If not, the agent drops the request and goes to recharge its batteries or refuel to the nearest station. After serving one request, the agent awaits for more requests until the simulation is finished.

Directory agent. SimFleet has a place where services provided during the simulation may be registered to be found later. This is done by the directory agent, which offers this subscription and search service. Usually, fleets are registered in this directory to be found by customers and to allow new transports to find them and sign up for the fleet.

Customer agents. Customers are the entities that want to perform an operation: calling a taxi to move from one place to another, send a package to a destination, etc. This entity is represented by the customer agent. In SimFleet customers do not have the ability to move. They call a transport service which goes to the customer's position, picks up the package (or customer in case of a taxi, a bus, etc.), and transports the goods to a destination. Customer agents depend completely on the transport agents. To get a transport service the customer looks for an appropriate fleet in the directory and contacts to its fleet manager to get a transport service for the customer. The fleet manager broadcasts the requests to some or all of their registered transports (depending on its strategy) and any transport interested in attending it will send a proposal to the customer, who has to accept or refuse it (depending on the customer's strategy too). The customer waits until the transport agent picks it up and, once they arrive at the destination, it stops its execution.

2.2 Related work

In this section, we describe the topics we have researched and the technologies we have applied during the development of this thesis, as well as some of the most relevant works that discuss them. This include notions about Planning, specifically Multi-Agent Planning (MAP) as well as Game-theoretic planning proposals. Finally, we will present Non-Cooperative MAP and how do we approach it for the design and implementation of our work.

2.2.1. Classical Single-Agent Planning

In the classical planning paradigm [7], the world is described through a finite set of *fully observable states*, which means that the planning agent has complete knowledge of the state's characteristics. The world is static and deterministic; it is not modified until an action is applied and, once it is, it changes to a single other state. During the planning process, external changes of the world are not considered. Lastly, the goals that have to be achieved are explicitly defined and immutable.

A state of the world is described by a finite set of *literals*. Literals are atoms composed of a predicate and a set of finite parameters, which can be empty, that represent objects of the world. For the world to change among states, *actions* must be applied. An action is composed by a set of *preconditions* (literals), which must be satisfied in the world state for the action to be applicable, and a set of *effects* that modify the world by deleting and/or adding literals.

The complete planning process can therefore be described as a search for an ordered set of actions which modify the world from its initial state to a state where all goals are achieved.

2.2.2. Multi-Agent Planning

In contrast with single-agent planning, where the world is only modified by the actions of one planning entity, Multi-Agent Planning (MAP) [8] takes place in domains in which many agents plan and carry out actions together in a shared world. The agents usually cooperate to achieve common global goals, which describes a *cooperative MAP*. However, if the agents are rational and self-interested, they will take into account both the global objectives and their own private interests when planning. In the latter case, the aim is to find a joint plan that completes all common goals while satisfying the agent's own concerns. We are then speaking about *non-cooperative MAP*.

Cooperative MAP is used to solve tasks that either cannot be solved by a single agent or would be better solved by many of them cooperating [9]. Such tasks tend to involve spatially and/or functionally distributed (different abilities) agents, which need from one another to solve the planning task [10]. Research in cooperative MAP has also been conducted from the Multi-Agent Systems (MAS) field, being mainly motivated by the distributed aspect of their tasks and systems, focusing on the design of planning coordination strategies for decentralized agents. Nevertheless, *decentralized planning* is essentially fixated with coordinating agent plans obtained from independent planning, so agents do not explicitly cooperate to solve their tasks. Neither cooperative MAP nor decentralized planning are able to solve tasks in scenarios where the self-interest of the agents must be taken into account.

2.2.3. Game-theoretic Planning proposals

Game theory is the mathematical study of the interaction between *rational self-interested agents*. [11, 12, 13]. A rational self interested agent presents its own description of states of the world that it likes, being its actions motivated by such description [14]. The preservation of the individuals' self-interests is crucial in game theory. We can differentiate between two types of game theory: Coalitional or Cooperative Game Theory (CGT), devoted to the study of agents interactions when forming coalitions [15]; and Non-Cooperative Game Theory (NCGT), focused on strategic equilibrium and the maximization of the utility of individual agents against the actions of other players.

Inside NCGT, we can categorize games as *strictly competitive* and *non-strictly competitive*. Strictly competitive games face agents with completely opposite interests. Therefore, an agent achieving its objective implies the defeat of all the other participants. These games are called zero-sum games [11], and the most direct example of them are two-player board games, such as Chess, in which the outcome is a win-loss situation. On the other hand, in non-strictly competitive games, agents do not explicitly aim to defeat or hurt other players. Instead, they have personal interest that may be in concordance or opposition to other agent's interests. These games are referred to as non-zero-sum games, also known as general-sum games, in which win-win situations can arise [16, 14].

From a planning perspective, we differentiate among *coalitional MAP* for CGT, *adversarial MAP* for strictly competitive agents and, finally, *non-cooperative MAP* for non-strictly competitive planning agents. Our work falls within the latter category. Following, we explain more details about non-cooperative MAP and how it is adapted to the objectives of this thesis.

Non-cooperative MAP

In non-cooperative MAP tasks, agents do not intend to harm each other, neither seek each other for help. Instead, as they have parallel and opposite interests, their goal is to seek an outcome that satisfies all participants through collaborative strategies and conflict resolution techniques. In this subsection, we illustrate the type of problem we aim to solve with a general-sum game example [12]. In addition, we show the matching between agents' strategies in a game and agents' plans in a planning task. Finally, we comment some non-cooperative MAP applications.

The Battle of the Sexes is a popular coordination game. We have defined the following instance, presented in the payoff matrix of Table 2.1 [14]. There are two players: the row and the column player. The first number of each cell represents the utility reported to the row player whereas the second is the utility reported to the column player. In our example, the players have to decide which genre of movie to watch; a Sci-fi movie or a Comedy movie. Each player has its own genre preference, being Sci-fi for the row player and Comedy for the column player. Nevertheless, both of them would rather watch a movie together than by their own, which is shown in the utilities of Table 2.1 that are 0 when each player chooses a different genre and above 0 when they select the same. This represents a general-sum game in which the self-interested agents will coordinate their strategies to obtain as much utility as possible.

	Sci-fi	Comedy
Sci-fi	**2, 1**	0, 0
Comedy	0, 0	**1, 2**

Table 2.1: Battle of the Sexes game in normal-form

If we compare the players' strategies to plans, we understand that a *conflict* exists when the players watch different movies, having both a utility of 0. In contrast, when the players agree on the same genre and one of them obtains less utility, we assume that it is because its plan is more expensive to execute. There are two unique Nash Equilibria (NE) [17] that can be derived from this game; the pair of strategies *Sci-fi, Sci-fi*, reporting a utility of (2,1) and the pair of strategies *Comedy, Comedy*, reporting a utility of (1,2). These are the only stable solutions from which the agents would not deviate, since any other solution would imply a loss of utility.

Non-cooperative MAP has been used for goal allocation [18]. In these settings, agents look for the optimal solution of a MAP task, which is their common goal, while contenting their own incentives. The aim is to divide the goals of the MAP task among the agents in a way that guarantees an optimal solution which maximizes the sum of the agent's utilities, a concept known as utilitarian social welfare.

Another use for MAP with self-interested agents is the coordination for problems in which many agents interact. In this line, there have been works which apply a pre-planning coordination, giving agents sub-taks derived from the decomposition of a common global task, so that they solve them individually [19]. In [20] a carsharing application is presented. This technique is used to find routes that travelers can share to save costs.

The works in [21] and [22] use a two-game approach to synthesize a stable joint plan that solves the MAP task in which the agents have a limited set of precomputed plans that solve their goals. Although these systems provide and guarantee certain properties like Nash equilibrium, Pareto optimal and fair solutions, there is a significant limitation to compute solutions in large problems.

Best-Response Planning (BRP) [23] is a proposal particularly devoted to solving *congestion games* [24]. Congestion games feature resources which, if used by simultaneous agents create a congestion; i.e: an increase in the agent's costs. BRP works as follows: First, an initial conflict-free joint plan is computed. Then, the so-called best-response dynamics take place; an iterative process in which agents propose, in turns, their best plan taking into account other agents plans, improving the initial solution. Non-cooperative MAP can therefore be seen as the coordination of agents' plans to achieve an executable joint plan. However, because of their private interests, such joint plan must be an equilibrium, a stable solution from which no agent is incentivized to deviate.

In [25], authors introduce the Better-Response Planning Strategy (BRPS), a modification of the BRP [23] which considers congestions and planning conflicts as a part of the agents' utility function, achieving a more realistic approach. In their model, the need for an initial conflic-free joint plan is avoided, since it generates synergies between agents that do not make sense in a non-cooperative environment. Finally, they study under which conditions the convergence to an equilibrium is guaranteed. Our work is highly inspired by the BRPS model and its authors' research, since we apply their principles to a real-world urban traffic domain.

As mentioned above, non-cooperative MAP problems can be mirrored by an equivalent non-cooperative game, in which the strategies that players follow are plans. This, however, makes the computation of equilibrium solutions harder, since planning is a computationally hard task. In our work, we develop this idea to redefine multi-agent simulations as multi-agent planning tasks performed by self-interested agents. To deal with the planning's computational hardness, we designed our own ad-hoc planner, which obtains optimal plans rapidly making use of the domain knowledge to reduce time and memory consumption. The plans it obtains are then used as strategies in a BRPS-like [25] process that obtains a stable solution (joint plan) for the simulation.

CHAPTER 3
Proposal

Our work revolves around SimFleet and its potential to aid in the improvement of urban traffic, providing accurate simulations of urban fleets which can be used both for research and testing. However, the current version of SimFleet has some limitations which we encountered while working on it. During the development of this thesis we have enhanced SimFleet by designing and implementing different modules which expand its functionalities. Besides that, we also explored the introduction of rational, self-interested agents to SimFleet's simulations. In this way, the private interests of transport agents could be taken into account, achieving more realistic simulations of taxi agents or electric autonomous vehicles (EAVs). For this, we developed a game-theory-based planning system used to decide the actions of every transport agent in accordance to their interests while, at the same time, guaranteeing the simulation execution by avoiding any conflict between them.

In contrast to other modules, the aforementioned system poses a major research problem, which we tackled from a scientific perspective. However, after the theoretical research we have also implemented the system, designing it for its integration with SimFleet. For this, we consider it the main contribution of this thesis, being, without a doubt, the most interesting and complex module among the developed ones.

In this chapter, we present all of the modules, describing their motivation and how they contribute to the improvement of SimFleet. We first describe the modules individually, giving later an overview of the whole system.

The topic of research of our thesis as well as the work we describe is framed in a research project promoted by the Ministry of Economic Affairs and Digital Transformation of Spain[1] which focuses on intelligent and sustainable mobility supported by multi-agent systems.

Following, we introduce each one of the developed modules, which are later explained in detail:

- **Load Generators**: Automatic simulation generators which aid in the creation of more complex and realistic simulation scenarios.

[1]*Ministerio de Asuntos Económicos y Transformación Digital*: https://www.mineco.gob.es/portal/site/mineco/

- **Free-floating Carsharing system**: Enhancement of SimFleet with the possibility of simulating a free-floating carsharing system, which required a new design of SimFleet's agents as well as their behaviors and interactions.
- **Best-Response Planning Strategy**: Simulation solver with rational, self-interested transport agents based on game-theory principles. Simulations are redefined as a Multi-Agent Planning task which is solved by a game, achieving stable solutions which preserve the agent's private interests.

3.1 Load Generators

One of SimFleet's main disadvantage arises when defining a simulation. A simulation is described in SimFleet by a JSON configuration file, which has to be manually written, including all agents and their attributes. This becomes specially troubling when trying to define big simulations, with a great number of agents.

To solve such issue, we developed two so-called generators [26], programs which, given a series of parameters, fill or create a new SimFleet configuration file, leaving it ready for execution. Besides that, the generators create more realistic simulations and agent distributions, being able to use cadastral information to obtain more accurate scenarios.

Firstly, we developed the *Charging station generator*, which locates the specified number of charging stations in the urban area where the simulation takes place. With it, various distributions of charging stations can be tested, seeing which ones achieve better traffic flow in the city.

Secondly, we created a *Load generator* of movements in the city. It locates customer and/or transport agents in the urban area and creates routes for them; i.e: defines their spawning and destination points (destination only for customers). The movement of agents it creates can be random or informed by real-city data (extracted from open data portals) which make the routes of customer agents more realistic by choosing origin and destination points according to *population density*, *traffic information* and *twitter activity*. The influence of each of these parameters is pondered by weights that are assigned to them.

3.2 Free-floating Carsharing System

Another of SimFleet's limitation was the way in which it defines urban fleets. Although SimFleet can model fleets of different types, the interaction between transport and customer always follows the same paradigm. Whether it is a taxi fleet picking up customers or a delivery fleet picking up and dropping packages, the customer agent is always the passive entity and the transport the active, performing all movement within the urban area. This was a problem when we started researching not so conventional urban traffic solutions, such as carsharing systems.

Being every time more concerned about environmental issues, some cities are restricting the traffic around their city centers to electric and low-emission vehicles only. To palliate this issue while still providing citizens with the ability to do private trips, carsharing systems can be used. This type of system presents a fleet of vehicles, which are parked in certain locations of the urban area and can be booked for their use by system users.

Wanting to do simulations of carsharing fleets, we developed a modified version of SimFleet that implemented a *Free-floating carsharing system* [27], which has the particularity that vehicles can be parked anywhere inside a predefined urban area. For that, we redefined from scratch the behavior of SimFleet agents. Customer agents were given the active role through the ability to book vehicles and walk to them. Transport agents were implemented to accept or deny bookings as well as driving customers to their desired locations. Finally, the *FleetManager* agent acted as the application through which system users could check available vehicles and their positions, and issue the booking.

The flow of the simulation execution was also changed, defining new situations in which the simulation would be considered finished. Typically, a simulation would only finish once every customer has reached their destination. In free-floating carsharing systems, however, it may happen that a certain customer does not find a close-enough vehicle available for booking, thus not being able to reach their destination. To define which vehicles are withing the walking reach from the customer, we introduced a new attribute, the *maximum walking distance*, which indicates how many meters a customer is willing to walk to reach their destination. This stops customers from booking vehicles which are so far away that it does not make sense for them to use. If a customer is unable to book a vehicle, the system will take that into account and stop the simulation indicating that it is uncompleted. This system allowed us to test vehicle and customer distributions (with the aid of the generators) to determine adequate locations and amounts of cars in carsharing fleets for different cities.

3.3 Best-Response Planning Strategy

Ultimately, the last SimFleet module and main contribution of our work is motivated, not by a limitation, but by the desire to explore self-interested agent coordination by means of game theory techniques rather than global scheduling or planning. In this way, we aim to obtain a simulation execution which preserves the individuality of the agents, not imposing any action, but allowing them to determine their actions by themselves and, at the same time, avoid conflicts with the rest of the agents of the scenario. For this, a completely new approach for carrying out the simulations is used, redefining SimFleet's multi-agent system and its agents interactions as a Multi-Agent Planning (MAP) [8] task.

Currently, coordination among SimFleet agent's depends on their own *strategies* as well as the interaction between them, usually performed by message passing. When designing such strategies, the user must take into account all sources for possible conflict and manage them properly to ensure a correct simulation execution. According to how the agents interactions are played out, the obtained solution (understanding solution as a correctly executed simulation process) may or may not preserve agent's individuality and self-interest. For instance, the FleetManager agent can be seen as a global scheduler which decides how to distribute travel requests among the transport agents of its fleet. However, depending on its strategy, it can also function simply as a travel request broadcaster, allowing the transport and customer agents to negotiate, reach agreements and avoid conflicts by themselves. The default transport agents of SimFleet have no other goal than serving customer requests. They feature no private interest, always accepting to serve a request without taking into account customer location.

Aiming to introduce the concept of rational, self-interested agents to SimFleet, we propose an extension which given a simulation scenario, redefines it as a non-cooperative MAP task, whose final goal is to solve the simulation by serving the travel requests of every customer in the scenario. In order to do that, an equivalent congestion game [24] is created. SimFleet's transport agents act as players whose strategies are plans. The plans are obtained by our own ad-hoc planner (Chapter 6) which is able to plan the agent's individual actions according to their interests while also taking into account the plans of every other agent, obtaining always a plan that is a best response to all other agents' strategies. The game is developed by a Best-Response Planning Strategy (BRPS) process (Chapter 7), in which the agents propose different strategies (plans), always in best response, improving their initial plans to avoid conflicts with other agents maximizing at the same time their utility. The process converges to an executable solution (joint plan) which is a Pure Nash Equilibrium [17], guaranteeing that no agent will deviate from it (change its strategy). In this way, the BRPS obtains a solution which indirectly achieves the global goals of the simulation (all customer travel requests are served) by capitalizing on the agents' own incentive to maximize its benefits, which the agent does by serving as many customers as possible. As the global goals are satisfied, the obtained joint plan is also a solution for the simulation, and can be executed as such.

3.4 Overview

As we have discussed on this chapter, SimFleet is a powerful simulation tool which has, however, a wide margin for improvements. Our work aims to enhance it to make it an even more useful tool that allows its users to experiment over more complex and complete scenarios.

During our research, we identified the need of a system for the automatic generation of simulation scenarios, which also allowed the user to test different distributions of charging stations on a city, locate vehicles and people who are in the city with particular goals depending on the problem and create their associated traffic flow. With it, new opportunities arise to simulate different types of configurations, which can be very useful for the research community and even public organisms like city halls that want to test the efficacy of charging stations (or any other of SimFleet's elements) in their towns.

Although the original version of SimFleet contemplates the possibility of vehicles carrying people or packages, other scenarios such as carsharing systems, which are very popular in recent years, are not taken into account. It could be extremely useful to have a tool in which to simulate different scenarios, being able to manage the quantity and distributions of the vehicles in any city. All this, making use of the aforementioned simulations generator too.

Finally, we had a need to simulate fleets of vehicles in which each of the individual vehicle has its own interest, such as taxis picking up clients, in realistic environments. For that, game-theory presented an adequate way to respect agent's self-interests. On the other hand, as agents need to know what to do and which are their best options, we had to introduce planning to consider every possible alternative. All of this made us came to the realization that a game-theoretic system with planning was needed, which is the main contribution of this thesis. Best-response dynamics, an iterative process in which agents propose their strategies always as a best response to all other agent's strategies; was used for its ability to obtain equilibria, together with an ad-hoc planner, designed exclusively for our domain, which exploits the seach space in a reasonable time given the restrictions of our modeling and other agents' contexts, which reduce the seach space up to a point where we can obtain optimal plans.

An overview of the final system can be seen in the graph of Figure 3.1, where every module is connected to other modules with which it interacts. As it can be seen, the generators' module and the Best-Response Planning Strategy (BRPS) and plannner modules are completely external to SimFleet and can run simply from a SimFleet configuration file. The free-floating carsharing module, on the other hand, is integrated in SimFleet's architecture. Altogether, we have achieved a system which is prepared to simulate many different types of urban traffic fleets with different agents and features. Coming up next, we describe in detail each of the modules. Specifically, we present the Load Generators on Chapter 4, the Free-floating carsharing system on Chapter 5 and the game-theoretic system, with the planner and the Best-Response Planning Strategy, on Chapters 6 and 7, respectively.

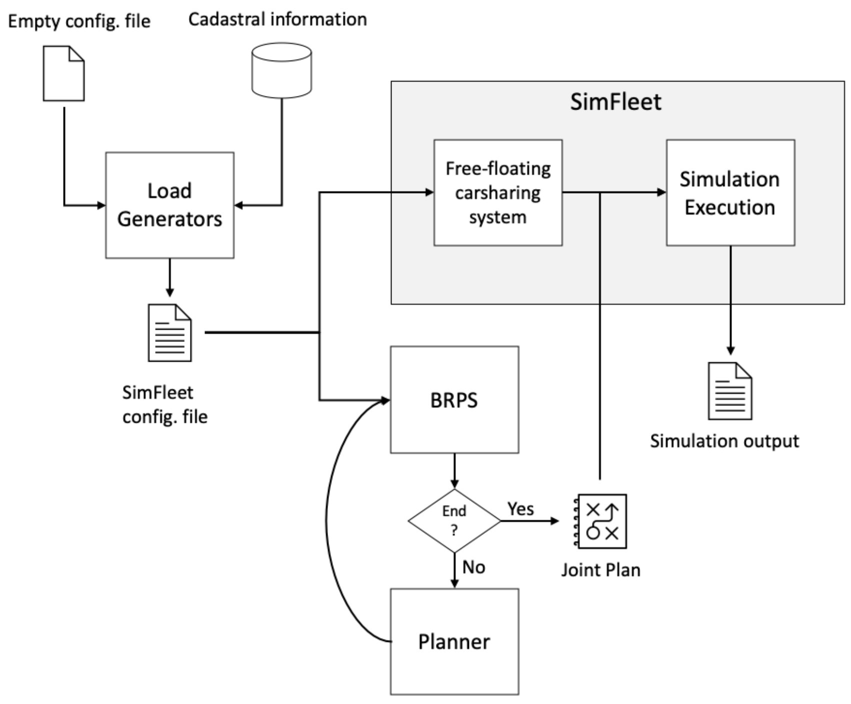


Figure 3.1: Proposal architecture

CHAPTER 4
Load Generators

For the development of our thesis we use SimFleet simulator, which is able to place different varieties of agents with custom behaviors over real-world cities to develop and test any type of strategies. Nevertheless, there is an important flaw in SimFleet to prepare simulations with a significant number of agents, and also to have an appropriate representation of the traffic of a city. Therefore, in this chapter, we propose a significant improvement for SimFleet: the inclusion of two generators at different levels. On the one hand, a charging stations generator to create several distributions of these infrastructures, and to be able to make comparisons and simulations with well-informed charging stations emplacing systems such as the ones in [28] and [29], that use several data sources to feed a genetic algorithm that obtains solutions. On the other hand, a load generator of agents on the move in a city such as urban fleets of taxis, private vehicles, delivery transports, buses, etc.; and customers of taxis or packages. Furthermore, this load generator can consider real data of the city, which implies a more informed approach to generate the real traffic of a city to be used in dynamic simulations.

A simulation on SimFleet is defined by its configuration file, which has to be manually written. Therefore, to include a large number of agents requires the manual definition of all of their parameters. These generators will allow SimFleet users to create realistic scenarios easily without having to write long configuration files, and more importantly, to generate load representing real traffic of the specific city by using available data such as population, traffic, and tweets, from open data portals[1], or gathered with other tools such as [30].

To automatically create experiments means to generate new configuration files or fill a previous one with data according to some parameters. For this, all generators include the option to get as input a configuration file and they are prepared to leave the present objects of such file unchanged while including the ones generated. Besides that, each generator works with a GeoJSON[2] file that defines the area of the real world where our simulation will take place and thus they have to populate. We will call this file the **city map**, since, usually, simulations are performed within the borders of a city. Since it represents a real-world location, all geometries defined in the city map are indicated with latitude-longitude points; to manipulate them we use the Python Shapely library[3].

[1] http://gobiernoabierto.valencia.es/en/data/
[2] https://geojson.org/
[3] https://pypi.org/project/Shapely/

4.1 Charging stations generator

The charging stations generator is used in order to test different distributions of charging (or petrol) stations of any kind over an area. The generator has many parameters; we will only present the relevant ones: n charging stations to distribute; p charging poles of the distribution; and distribution type, $\{random, uniform, radial\}$, that affects how stations are positioned inside the area.

Each station may have several charging poles. The allocation of charging poles between stations will be discussed further on. The generator outputs a file in GeoJSON format which indicates the type and position of each station. Such a position, however, can not be any point inside the city map but rather a valid point: a point belonging to a street or road. For obtaining valid positions, we make use of the function *getValidPoint*, which given a point returns the nearest valid point to it. This function uses the service *nearest* of OSRM, which returns the nearest point belonging to a street or road with respect to the specified coordinates.

4.1.1. Random distribution

In this type of distribution, n valid points are generated within the city map. Each of the points indicates the position of a charging station. For the generation of points, the bounds of the polygon that defines the city map are used: x_{min}, $y_{min}, x_{max}, y_{max}$. Using these values, random x and y coordinates are generated for each point. Then, the corresponding valid point is obtained. If the point is contained in the city map, it is stored as a station location; otherwise, it is discarded. This process is repeated until there are as many stored points as the number of stations.

A random distribution is interesting from the experimentation perspective to compare other more informed distributions against it.

4.1.2. Uniform distribution

This distribution divides uniformly[4] the city map (see Figure 4.1a) into equal size cells, generally with rectangular shape. To do so, it creates a wider working area, the *grid* (Figure 4.1b), defined from the bounds of the polygon representing the city map. If such bounds are x_{min}, y_{min}, x_{max}, y_{max}, the grid is the four vertex polygon defined by the points $\{(x_{min}, y_{min}), (x_{min}, y_{max}), (x_{max}, y_{max}), (x_{max}, y_{min})\}$.

The number of rows and columns of the grid is determined by the number of stations (n) and its width and height, following the criteria shown in Equation (4.1.1). If the number of stations is a perfect square, i.e., its square root is a positive integer, the grid will have the same amount of rows and columns, obtaining exactly n cells. However, in general, the number of rows and columns will de-

[4]The name "Uniform distribution" does not refer to a probability distribution but to how stations are divided in the city map.

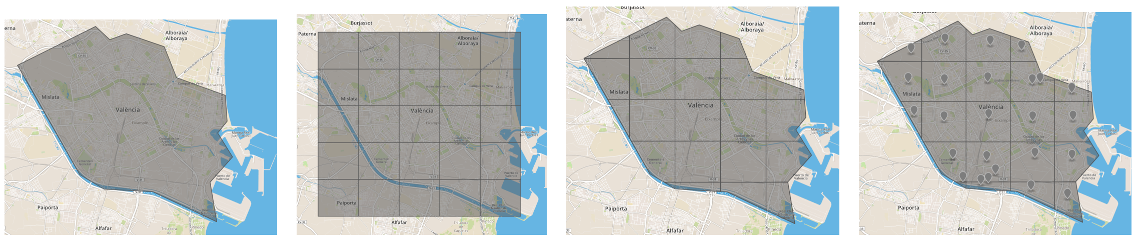
pend on whether the grid is wider or higher, having one more column than the number of rows (or vice versa) accordingly.

$$\begin{cases} rows = cols = \sqrt{n} & n \text{ is perfect square} \\ rows = \lfloor \sqrt{n} \rfloor, cols = \lceil \sqrt{n} \rceil & height < width \\ rows = \lceil \sqrt{n} \rceil, cols = \lfloor \sqrt{n} \rfloor & otherwise \end{cases} \tag{4.1.1}$$

Once the grid is split, we trim each of the cells against the city map, which causes cells outside of it to disappear and those laying over its borders to get an irregular shape (see Figure 4.1c). These final cells are stored in a list of valid polygons where stations can be placed.

Next, an iterative process begins to allocate every station. The list of valid polygons is traversed and a station is placed in the closest valid point (using the *getValidPoint* function) to the centroid of the polygon. This process finishes once every polygon has one station in it or all stations have been allocated. After this, if there were still stations to allocate, the rest are positioned in a random valid point of a randomly chosen valid polygon (Figure 4.1d shows a possible outcome).

Besides this, we also implemented a random version of this distribution in which the city map is divided in the same way emplacing, however, each station in a random valid point within a randomly selected cell.



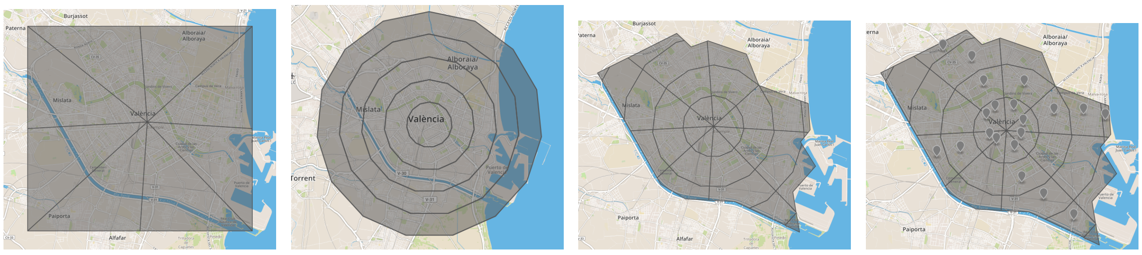
(a) City map **(b)** Working area or Grid **(c)** Grid trimmed against city map **(d)** Cells populated with stations

Figure 4.1: Uniform distribution of stations process

4.1.3. Radial distribution

This distribution was inspired by the radial distribution of activity within certain cities, which presents a higher rate towards its core and decreases as it gets closer to the peripheries. It requires an additional parameter c, which indicates the number of circles that will be used to divide the city map. The division process begins by defining two copies of a wider working area, created as described for the uniform distribution. One of those copies will be divided into a series of triangles, 8 by default as can be seen in Figure 4.2a, by joining every vertex and sides' middle point with the centroid of the original city map. As for the other, it will be divided by c concentric circles with an initial radius r calculated taking into account the map dimensions. To avoid circle overlap, each circle is trimmed against the one created before it, starting by the last (and biggest) created. This obtains an area with a middle circle and many rings around it, as can be seen in

Figure 4.2b. Take into account that from now on when we mention circles we will also be referring to the rings. Next, the two aforementioned areas are intersected with each other, dividing each circle into 8 polygons, and trimmed against the city map, obtaining a division of it as shown in Figure 4.2c.



(a) Triangle division of working area **(b)** 5 circle division of working area **(c)** Final city map division **(d)** 20 stations between 5 circles

Figure 4.2: Radial city map division

Each station is then allocated in the closest valid point to the centroid of one of the polygons. The number of stations per circle (n/c), as well as the amount of polygons a circle has, is taken into account to allocate the stations as uniformly as possible within the city map and each circle. The algorithm populates each triangle starting from the inner circle and moving towards the outer. Once all polygons of a triangle have a station assigned, the algorithm will select the next triangle according to the number of stations and the total number of polygons to divide the stations uniformly within a circle. The final distribution can be seen in Figure 4.2d.

The number of stations may be greater than the number of polygons since the number of circles is determined by the user. The algorithm described above places only one station in each polygon. In this case, the rest of the stations are assigned a random position by randomly choosing a polygon and a valid point within it.

We also implemented a fully random version of this distribution that divides the city map in the same way, takes into account the number of stations per circle, but chooses randomly the polygons within a circle as well as a valid point within them.

4.1.4. Charging poles allocation

The amount of charging poles (spots for a vehicle to charge) we want in our configuration is one of the parameters of the station generator. There must be at least one charging pole in each station. After this, if there are more charging poles to be distributed they will be allocated according to one of the following methods:

The first one distributes the points evenly by traversing the list of stations, adding one point to each until all points have been allocated. Then, the list of stations is shuffled, so as not to benefit stations that are traversed first. In this way, the stations will have either p/n or $p/n - 1$ charging poles.

The alternative is a pseudo-random distribution of the remaining points that works choosing a random amount of points and a random station to which assign them. To avoid a too uneven distribution of poles, such a random amount can be limited by a parameter that indicates the maximum percentage of the total charging poles that a single station can have. For instance, using a maximum percentage of 30%, we ensure that no station will have more than $0.3 \cdot p$ charging poles.

4.2 Load generator of movements in a city

The load generator is used to create either a random or informed load on the simulation. Such load can be adapted to any of the agent types that SimFleet offers: electric vehicles, taxi fleets, customers for taxis, delivery vehicles, packages, etc. The relevant parameters of this generator are: agent type t; amount of agents n; minimum distance min_dist in meters; starting delay d in seconds; amount of agents per batch $agents\_per\_batch$.

The generator aims to create a movement of at least min_dist meters of n agents of t type within the borders of a given city map. The delay parameter d determines at what time of the simulation the generated agents will start running; by default, it is 0. The amount of agents per batch is introduced to give different delays to sets of $agents\_per\_batch$ agents, which will begin its execution at the same time. This may be useful when generating a large number of agents. If indicated, the first batch of agents will have a delay of d; the second, a delay of $2d$, and so on.

As we mentioned above, all generators are prepared to receive an existing SimFleet configuration file as input and fill it with agent data. This enables the use of the load generator to introduce, in the same simulation, different types of agents in various amounts, with different delays and batch sizes, to create a complex system.

4.2.1. Random movement generator

The random load is created by choosing a random route (random origin and destination points) for the agent to perform. Both random origin and destination points must be valid points of the area, and they must be at least min_dist apart from one another. This process is repeated to create n agents of type t. The origin point will determine where in the area the agent will spawn, whereas the destination point indicates where it will finish its execution. If the agent type is customer or package, the movement is performed by the corresponding transport vehicle that carries it after it gets picked up.

4.2.2. Informed movement generator

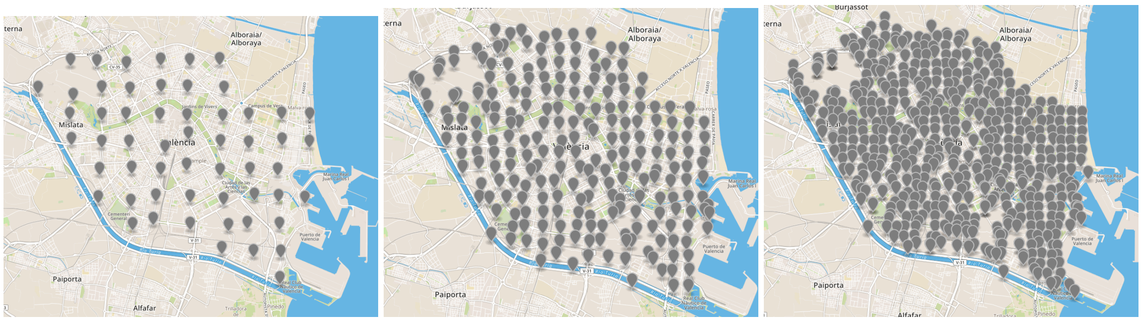
The informed version of the load generator aims to reproduce more realistic movements around the city map. For this, it is necessary to provide the generator with relevant data from which to ground the routes of the agents. This data can be obtained from diverse sources; often open data platforms that the government of a city or country makes accessible for its citizens. For our generator, we used the following data:

- **Population information**: It shows the amount of people that live in different zones of a city. The population information (P) is defined as: $P = \{(C_1, p_1), (C_2, p_2), \dots, (C_n, p_n)\}$, where C_i is a closed polygon representing a zone in the city together with its population p_i.

- **Traffic information**: It shows the number of vehicles moving around a certain area. The traffic information (T) is defined as: $T = \{(R_1, t_1), (R_2, t_2), \dots, (R_n, t_n)\}$, where R_i is a polyline that follows a street or road indicating the volume of traffic t_i.

- **Twitter activity**: Information about the amount of geo-located tweets, from the social network Twitter, tweeted from a certain location. This information can be used to determine where a representative percentage of the population is spending their time. The Twitter activity (A) is defined as: $A = \{(Q_1, a_1), (Q_2, a_2), \dots, (Q_n, a_n)\}$, where Q_i is a point represented as a latitude-longitude tuple and a_i the number of tweets in such coordinates.

The information is used to create a probability distribution among the available points of the area. The selection of the origin and destination points will be performed according to such distribution. For this, we begin by creating a set of available points. The city map (M) is divided as if it was a grid similarly as explained in Section 4.1.2 for the uniform distribution, obtaining $M = \{(G_1, O_1), (G_2, O_2), \dots, (G_n, O_n)\}$; where G_i is a closed polygon and O_i the nearest valid point to the centroid of G_i. The number of rows and columns of the grid is a configurable parameter and it determines the granularity of the system. A higher amount implies more cells in the grid which directly translates into more available points, as it can be seen in Figure 4.3. The more points, the more distributed will be the probability.



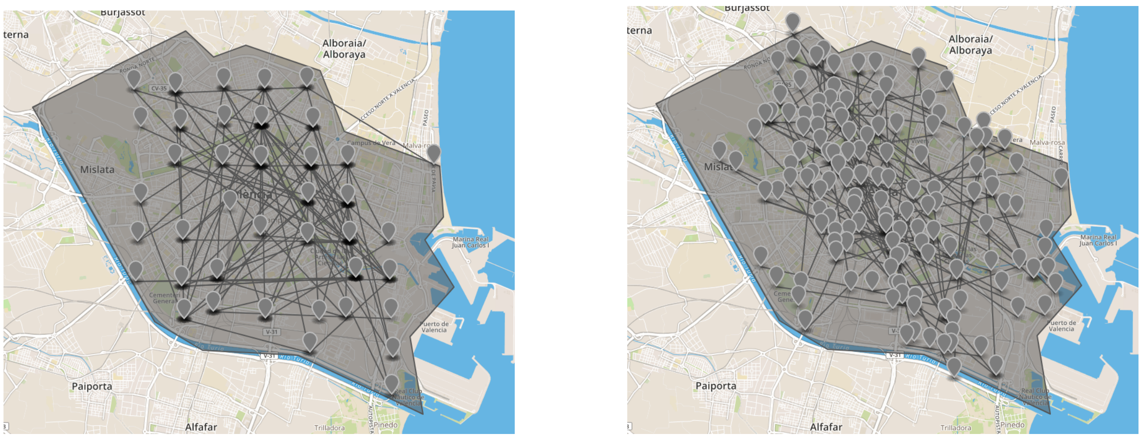
(a) 10 rows and cols **(b)** 20 rows and cols **(c)** 30 rows and cols

Figure 4.3: Number of available points according to map division granularity

By merging the city data with M, we join, for every polygon G_i, the population, traffic and Twitter activity amounts that take place within its area: $M = \{(G_1, O_1, \{p_1, t_1, a_1\}), (G_2, O_2, \{p_2, t_2, a_2\}), \dots, (G_n, O_n, \{p_n, t_n, a_n\})\}$ and calculate the probability associated to each point O_i as in Equation (4.2.1):

$$prob(O_i) = w_p \cdot \frac{p_i}{\sum_{j=1}^{N} p_j} + w_t \cdot \frac{t_i}{\sum_{j=1}^{N} t_j} + w_a \cdot \frac{a_i}{\sum_{j=1}^{N} a_j}; \text{ with } w_p + w_t + w_a = 1 \tag{4.2.1}$$

where w_p, w_t and w_a are weights that control the influence of each of the factors over the probability. Finally, the generator takes into account the set of available points (S) and their corresponding probability: $S = \{(O_1, p(O_1)), (O_2, p(O_2)), \ldots, (O_n, p(O_n))\}$ to generate the routes.



(a) 10x10 map **(b)** 30x30 map

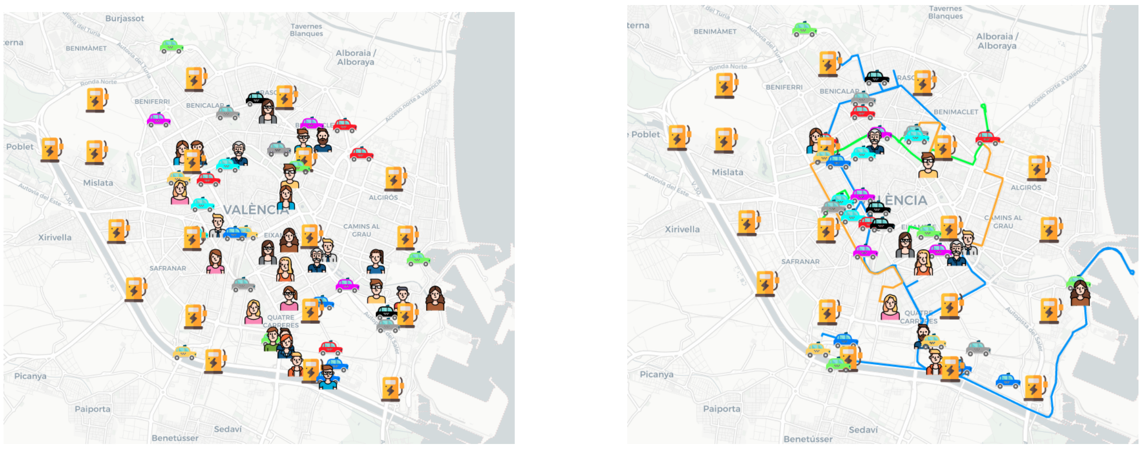
Figure 4.4: 100 routes example

Once every point in S has its probability assigned, a process to create the n routes begins (see examples of Figure 4.4). This process is very similar to the one used in the random load generator, but this time the origin and destination points are chosen from S according to the probability distribution and ensuring the min_dist between both points

4.3 Simulation example

In this section we show an example of a complex simulation scenario set up with the aid of both the charging station generator and the load generator.

The simulation defined takes place over the main area of the city of Valencia, Spain. We generated 20 electric charging stations distributed uniformly in the area. As for the load (agent movement in the city), we used the informed load generator to create a fleet of 30 electric taxis and 30 customers, with a granularity of 30. Taxis were created with random values for their autonomy so that some of the taxis would eventually need to charge in one of the stations. As for the customers, the weights of Equation (4.2.1) were $w_p = 5/12$, $w_t = 1/4$, $w_a = 1/3$, increasing the impact of population and Twitter activity. The routes of the customers were defined by points that were at least 700 meters apart. For the taxis creation, the weights of Equation (4.2.1) were $w_p = 1/3$, $w_t = 5/12$, $w_a = 1/4$, giving more importance to traffic and population. Figure 4.5 shows the described scenario as presented in SimFleet.



(a) Experiment setup **(b)** Experiment running

Figure 4.5: Experiment shown in SimFleet

As it can be seen, the generators allow us to define more complex and interesting simulation scenarios fairly easy. In cases like the described in this section, where we have open data of the city where the simulation takes place, we can also achieve more realistic modelings by basing transport allocation and customer movement on real data.

4.4 Chapter remarks

In this chapter we have identified the need for enhancing SimFleet simulation potential and presented two tools to do so: the charging stations generator and the load generator. Knowing that one of SimFleet's purposes is the implementation and comparison of agent strategies, the generators are effective to help in the research of solutions for traffic congestion or any other type of challenge derived from city sustainability. Provided we have access to city data, with the use of the informed load generator, we can test different driver behaviors over realistic settings to identify problem sources and look for appropriate solutions.

As for the electrical vehicle (EV) charging stations infrastructure, thanks to both generators we can simulate different distributions over a city and, using its real city data, recreate movement within it to analyze the performance of each distribution. This information can be of use for municipalities or other entities in charge of infrastructure creation.

The placement of a charging station should take into account current traffic trends but one must keep in mind that it may as well have an impact on traffic once the station is working. In future work, we aim to develop coordination strategies among transport and station agents in order to find ways of optimizing traffic and achieving a maximum global utility. Such strategies may be the basis for autonomous EV in the smart cities of the future.

CHAPTER 5

Free-floating Carsharing System

The use of privately owned vehicles in cities is becoming every time more inconvenient for the citizens. As the concerns about carbon dioxide emissions increase, cities adapt by creating more green areas, penalizing or completely banning the use of private vehicles in their city center, and encouraging both public transport and electric vehicles. Besides that, there are problems that are inherent for vehicle owners like the lack of parking space, which implies on many occasions the need for paying for a private space. To such disbursement, one must add fuel or electricity expenses as well as vehicle maintenance.

Considering all of these inconveniences, a good amount of the people that live in cities opt by not buying a vehicle. For such users, public transport usually fulfils their displacement needs. However, there are cities in which, because of their structures or simply by the lack of resources, the public transport services are not enough to comply with the needs of citizens. Besides that, there might be some specific trips which entail needs that can not be provided by the public transport system.

Aiming to solve all of the previously mentioned issues, carsharing systems were proposed [31]. In these systems, private vehicles are owned by an enterprise, which rents them temporarily to their customers. In general, the vehicles are parked in specific locations and must be either returned to the original location or parked in a different predetermined location. Also, carsharing companies are transitioning to the use of electric vehicles, since they offer great performance moving inside urban areas. According to [32], Europe accounts for about 50% of the global carsharing market and is expected to grow further to 15 million users by 2020.

Taking this into account, the system we will describe in this chapter is a modification over the original carsharing system: a free-floating carsharing system [33]. These systems are much more flexible for the users since their vehicles can be picked up and parked anywhere within a specified urban area. Customers, that have access to vehicle locations, book a vehicle for a determined amount of time. The company takes care of vehicle relocation and recharge if necessary. For a monthly fee, these companies provide the benefits of owning a private vehicle without its drawbacks.

Free-floating carsharing systems need to be tested and improved to provide better service. While testing on real cities might be expensive and sometimes completely impossible, the use of simulators can be of help. Our proposal is to design and implement a free-floating carsharing system for the simulation software SimFleet, hoping that it provides a platform to test a carsharing system over real city infrastructures.

5.1 System description

A free-floating carsharing system has a set of *customers* who will use a set of *transports* to travel. Transports will be located anywhere within the borders of a certain city or urban area. Customers know the location of available transports at any time and can issue a booking using, for example, a website or an application. After making a booking request, customers wait for confirmation. If their booking has been accepted, they move to the vehicle (transport) and can access it to travel anywhere. Once the customer has finished using the vehicle, they must leave it properly parked within the mentioned urban area.

There are some problems inherent to free-floating carsharing systems. Since vehicles can be parked anywhere, it may happen that at any given time some users find all the vehicles parked too far away to be able to use them. In an extreme case, the available vehicles may be even further away than their destination and therefore it would not make sense for them to use the service. A system of such flexibility requires careful attention to detect and resolve such situations. In real life, carsharing companies relocate their vehicles from time to time to ensure proper distribution.

Another problem is the recharging or refueling of the vehicle. Ideally, a customer should find the transport fully charged (refueled) or at least with enough energy (or fuel) to complete his journey. We understand that carsharing companies are aware of their vehicles charging and periodically charge (or refuel) those who need it, driving them to charging (or petrol) stations or simply charging (or refueling) them on-site with a generator (or deposit).

To address these issues, the authors in [34] present a "staff" agent who deals with the recharging of vehicles. In addition, customers only book vehicles that will be available within 30 minutes of the booking being issued. The aim of SimFleet, however, is the development and comparison of the agents' strategies. That is why we decided to give the above-mentioned problems very simple solutions that a future user can improve them to what he or she considers best.

On the one hand, a maximum walking distance for customers has been introduced which indicates how many meters they are willing to walk to get a vehicle. This parameter can be manually defined for each customer in the simulation or not specified to ignore the restriction. A customer with this defined value will not book or use transports that are further away than what the value indicates. This may result in some customers not making their journey if they do not have a vehicle within reach at any time, and hence, the simulation would end with some customers not having reached their destination. In this way, a SimFleet user can also evaluate if his or her initial vehicle distribution is suitable for a certain customer distribution, or even, it could be considered to make relocation of the vehicles in these cases. We must note that the check of the distance between a customer and the available vehicles is done by calculating the straight line distance between them. As it would be expected, the distance that the customer ends up covering is usually greater than this estimate, so, in some cases, customers will exceed maximum walking distance restriction, since the actual route from its origin to the vehicle is usually longer than the straight line distance. We use an optimistic estimate of the distance to avoid calculating the actual route to every available transport, which would overload the routing server in simulations with many agents and increase the simulation time.

On the other hand, to simplify the experiments performed in this work, we assume that the vehicles are recharged on site each time they finish a trip, i.e., it is assumed that the vehicle has its full autonomy for each new customer.

5.1.1. Agents

Fleet Manager. It acts as the "application" by which customers check available vehicles and their positions. For that, it maintains the updated information about the location and state of every transport in their fleet and sends this information to any customer that requests it.

Transports. These agents act as the vehicles of the system of any type; e-cars by default. Transports are registered to a fleet and managed by their Fleet Manager, whom they will periodically inform about their location and status. Once a booking request of any customer arrives to a transport, it must reply accepting or rejecting it depending on their state. Although ideally customers would only send requests to available transports, in a multi-agent system it may happen that a booking request arrives to the transport when it has already been booked by another customer, or even when it is being used. In such cases, the transport will reject the request. If the transport accepts the request it will change its status to "booked" and wait for its assigned customer to arrive. When the customer arrives, the transport allows him to enter and drives him to its desired location. Upon arriving at the trip destination, the customer finishes using the transport, making it available again.

Customers. Customer agents act as the carsharing system users. Every customer is in its origin position and it has a destination that must reach. For doing so, upon spawning, the customer asks for the available transports to the Fleet Manager. If there are no available transports, the customer would wait for a determined amount of time before asking the Fleet Manager again. Once the customer receives the transports information, it can make a booking request to the vehicle of his choice. If the request gets accepted, the customer will walk to the location of his booked vehicle and, once accessed to it, drive it to his destination. After completing the trip, the customer has achieved his goal and so it stops its execution. As we mentioned earlier, if the customer has a maximum walking distance defined, it will only book transports located within his reach.

The simulation will finish once all Customer agents have reached their destination or if the remaining Customer agents can not book any of the available cars, in which case the impossibility of completion of the simulation will be indicated in the output of the execution.

5.2 Design of intelligent strategies

SimFleet is coded in Python 3 and, as commented above, makes use of SPADE as a base for the implementation of its agents. A SPADE agent can have one or many *behaviors* that, upon execution, will define the actions of the agent. Besides that, a series of agent-specific methods can be defined for the agent and called by the behavior execution.

Agents in SimFleet's carsharing system have a *strategy behavior* that implements their way of interacting with the system and each other during the simulation. These strategies model how agents behave in a carsharing system by means of a finite-state machine that represents the states through which the agent travels and which implement the negotiation and decision-making algorithms to book a vehicle and use it.

The agent strategy behavior defines its interaction with the system and the other agents. Generally, they are derived from the SPADE class `CyclicBehaviour`, which is a behavior that keeps executing itself until its goal has been completed. To determine the actions of the agents, an attribute representing their *state* is used. Depending on its state, the agent will pay attention to certain interactions (messages) or simply ignore them. This state is usually changed by the strategy behavior upon receiving a certain interaction, but it can also be changed internally by the agent itself. Therefore, it is very important to manage the value of the state attribute carefully or the agent behavior may become unpredictable.

The agents' strategies are implemented as SPADE's `FSMBehaviour`, a behavior composed of a Finite-State Machine (FSM). In this way, we designed the strategies as a FSM and match every possible value of the agent's state attribute with a different state of its Strategy Behavior. Also, states for Transport and Customer agents were introduced to reflect their status in the carsharing system.

Next, we describe the Strategy Behavior of the three involved agents, focusing on the actions that take place in each state and the transitions.

Fleet Manager Strategy Behavior. The Fleet Manager agent is in an endless state awaiting for messages. It can receive two types of message: a Transport agent informing about its state (available or booked), or a Customer agent asking for the list of available transports. Every time a message from a transport arrives, the Fleet Manager updates its internal list of available transports. When a customer asks for it, the Fleet Manager replies only with the transports whose current state is free.

Transport Strategy Behavior. A Transport agent is initially waiting for booking requests. When it is booked, it waits for the customer to arrive at the vehicle and finally moves to the destination. A Transport agent can be in one of the following states: (1) waiting to be booked, (2) waiting for the customer to arrive, or (3) driving to the customer's destination. Transitions between states are shown in Figure 5.1.

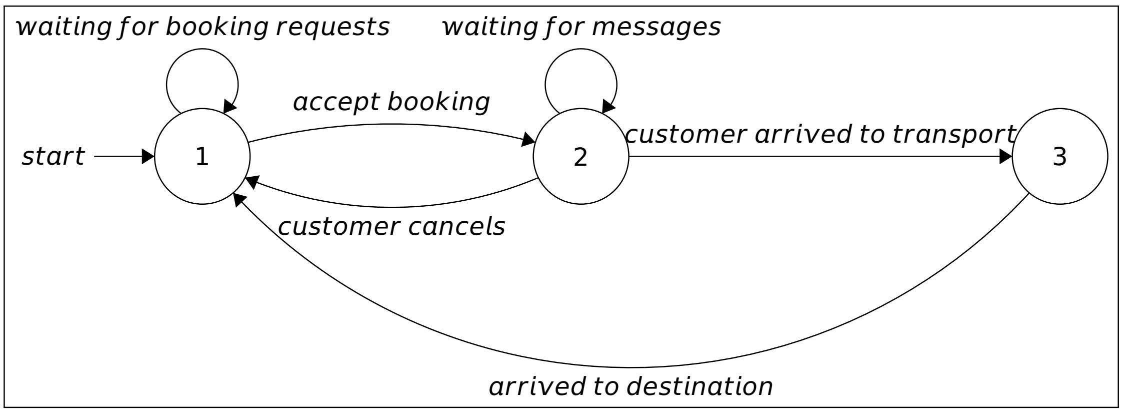


Figure 5.1: Transport Strategy Behaviour as a FSM

Customer Strategy Behaviour. A Customer agent can be in one of the following states: (1) making a booking, (2) waiting for the booking to be accepted, (3) walking to the booked transport's position, (4) inside the transport driving to his destination, or (5) in his destination. Transitions between states are shown in Figure 5.2.

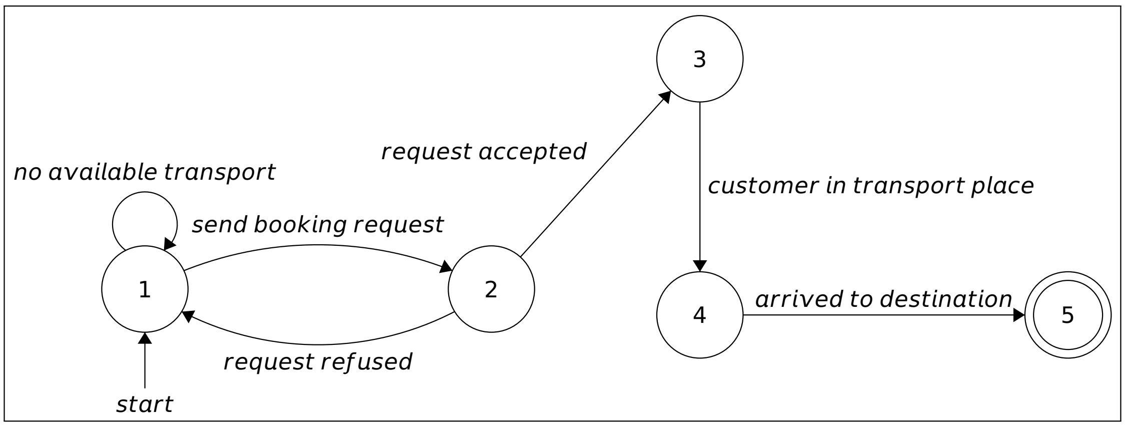


Figure 5.2: Customer Strategy Behaviour as a FSM

Besides the agents and their strategies, some other components of SimFleet were modified in order to be able to launch the application using our own agents and executing simulations. One of such modifications was the addition of Customer agent movement in the web application that acts as a User Interface for the visualization of the simulation.

5.3 Experimentation

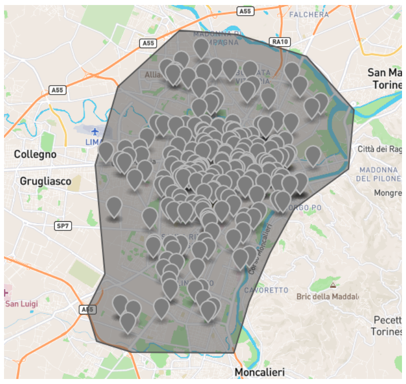
In this section, we show the flexibility of our SimFleet extension by creating and executing different simulations. For that, we make use of the simulation generators, presented in Chapter 4, which enable SimFleet users to easily generate scenarios based on real-world data, achieving more realistic simulations. As for the evaluation of the simulations, SimFleet includes many metrics like the time a customer is waiting to be picked up or the delivery time for delivery vehicles. For our system, we use the customer walking distance, in meters; i.e., the distance the customer agent moves from its origin point to their booked vehicle's position.

To run experiments as close to reality as possible, we based the location of the customer agents in a dataset containing origin and destination points of carsharing trips in the city of Turin, Italy, compiled over a period of two months [35]. The origin and destination points of the customer agents matched the ones in the dataset. Then, a certain number of transport agents were placed among the city area following different types of distributions: a *random* distribution, which simply locates vehicles at valid points inside the city area; a *uniform* distribution, which divides the city area as a grid and places a vehicle centered inside each cell; and finally, a *radial* distribution, which places more vehicles in the center of the city and reduces the number of them in the outer parts. Finally, concerning customer behavior, we assume that customers always book the closest available transport.

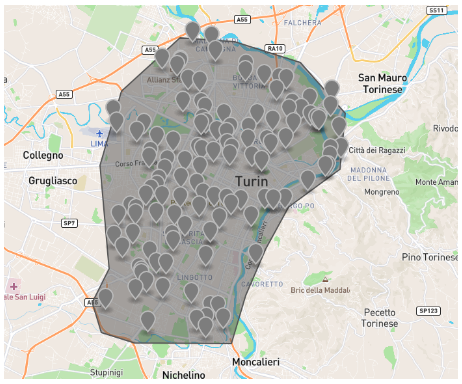
A graphic representation of the city area considered for the simulation can be seen in Figure 5.3a. The gray polygon encloses all of the points of the dataset, except the ones corresponding to the airport, which we did not consider as it is outside of the city. The points represent the origin positions of 250 customers (origin of carsharing trips in the dataset). In 5.3b, 5.3c and 5.3d random, uniform and radial distributions of 125 vehicles are presented, respectively.

We executed simulations with 250 customers, 125 or 250 transports distributed in one of the three aforementioned ways with maximum walking distances of 2000, 1500 and 1000 meters, and compared the mean customer walking distance and its standard deviation. The results can be seen in Table 5.1. Since the maximum walking distance restriction is checked over an optimistically computed distance, there are cases in which the real distance that the customer walks is considerably higher than the defined maximum. We considered these cases outliers and we do not show them since we understand that real customers would not walk such distances.

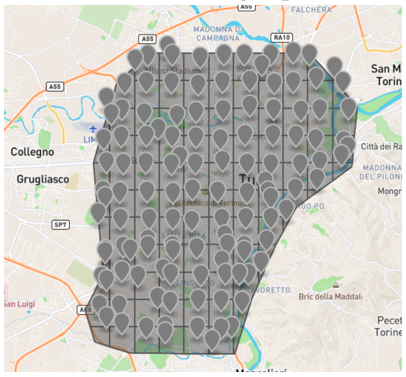
These experiments show how our proposal can be of use to free-floating carsharing managers to estimate, for instance, the most appropriate number of vehicles for their fleets, and how to distribute them through the city to improve customer experience by locating vehicles as close to them as possible. In this particular instance, the alternative vehicle distributions do not present any significant differences in terms of customer walking distance. Of course, system users can define and analyze any other metric they consider relevant.



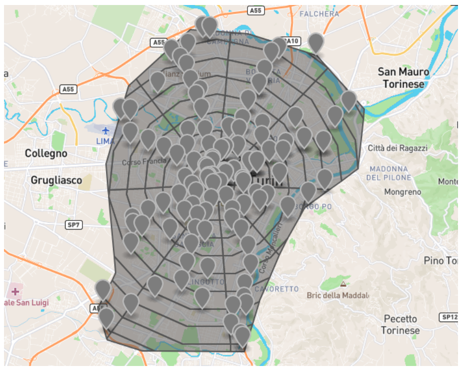
(a) Customer origin points



(b) Random distribution of vehicles



(c) Uniform distribution of vehicles



(d) Radial distribution of vehicles

Figure 5.3: Turin city area considered for the simulations

Distribution	# customers	# transports	Max. w.d. (m)	Mean w.d. (m)	σ (m)
random	250	125	2000	1019.70	715.37
uniform				1046.86	727.02
radial				1058.79	752.05
random			1500	953.58	639.12
uniform				928.30	631.50
radial				997.76	636.06
random			1000	799.37	494.27
uniform				830.46	537.28
radial				798	492.99
random		250	2000	967.82	728.07
uniform				944.21	718.36
radial				965.36	754.59
random			1500	886.08	608.45
uniform				886.65	706.79
radial				893.07	662.45
random			1000	759.97	519.40
uniform				769.65	506.39
radial				705.17	549.52

Table 5.1: Customer walking distance (w.d.) comparison with different simulation configurations

5.4 Chapter remarks

In this chapter, we have designed and implemented a free-floating carsharing system based on SimFleet, a multi-agent urban fleet simulator. With the presented system now integrated, SimFleet offers a large number of configuration options to simulate complex scenarios that reflect real-life fleet operations, giving it more potential to aid in solving urban traffic problems. The system can support fleets of many different types in the same simulation. For instance, we could simulate a taxi fleet together with a carsharing one in the same city. Consequently, the simulator is a great tool for evaluating transport and people distributions over actual cities and analyzing the effect of different agent strategies.

In future work, we will extend the system implementing other types of carsharing as well as different types of carsharing trips. Besides, we want to explore negotiation and coordination strategies among agents for a better resolution of complex simulation scenarios. Finally, we also consider improving the current system by adding the option of vehicle relocation during the simulation, used when customers have no access to any free vehicle because of their location

CHAPTER 6
SimFleet Planner

The work described in this chapter and the following Chapter 7 is motivated by the research on rational, self-interested agents. An agent with those features has its own private objectives which, in practice, translates to its unique utility function. Our goal was to introduce such agents to SimFleet's simulations to turn transport agents into electric autonomous vehicles (EAV). Such vehicles may belong to a taxi fleet thus having the interest of serving customers' travel requests, getting compensated by it. By introducing EAVs from different taxi companies, for instance, we create a competitive scenario where agents compete to serve as many customers as possible. However, since the aim of SimFleet is to simulate, we had to ensure that the aforementioned scenario was solvable, avoiding the conflicts that generally arise between agents in this type of context. For that, we model the simulation as a multi-agent planning task (MAP), specifically a non-cooperative MAP task; one in which agents do not create alliances to solve the global goal of the task but rather the task is solved by coordinating the way in which agents solve their own private goals.

With this modeling, the global goal would be to serve all customer agents, thus finishing the simulation. On the other hand, the agents will aim to maximize their utility, serving as many customers as possible. To avoid conflicts, the actions of each agent are decided by a game-theoretic process: A Best-Response Planning Strategy (BRPS) process, explained in Chapter 7. With it, transport agents play a game in which their strategies are plans; i.e: the list of actions they aspire to do in the scenario. During the development of the process, the agents propose plans, in turns, making sure they always are in best response to any other agent's plan. In other words, they ensure their actions are the ones that report them a higher utility with respect to all actions of other agents in the scenario. After a series of game iterations, the process converges to a stable solution or equilibrium; a solution from which no agent is incentivized to deviate. In this way, we achieve a joint plan (union of agent plans) which solves the global goals while preserving the agent's self-interests.

In this chapter we tackle the creation of individual agent plans. For that, we developed a planning algorithm which given a simulation scenario, defined by a SimFleet configuration file, and a transport agent in that scenario, builds the optimal plan for such agent, taking into account both the characteristics of the scenario as well every other transport agents plans, if any.

The planner is designed to be integrated in a BRPS process, being used by the agents to propose their best strategy in every iteration. Because of that, it expects certain elements upon its initialization which it uses to take into account the strategies of other participant agents. Nevertheless, it can work independently, although only for scenarios defined following the constrains of our urban traffic domain, which we also introduce in this chapter.

Following, we describe our urban mobility domain together with the transport agent's utility function and the conflict sources of a scenario that follows the domain constraints. Then, we introduce the actions that are doable in our domain and show the structure of the plans our algorithm obtains. After that, we describe in detail the planning procedure, how our planner searches for the optimal plan among all alternatives. For that, we present the different elements of the planner explaining their function in the planning. In addition, we comment on the planner performance and the techniques we used to enhance it. Finally, we develop a trace of the obtention of a plan in a small planning scenario which serves as example to illustrate the functioning of our algorithm.

6.1 Urban Mobility domain

Our urban mobility scenarios model a real-world smart-city urban area. Within the urban area there can be charging stations in determined locations and customers, which have an initial position and a travel request assigned to them. Additionally, there are EAV, to which we will refer as transport agents or simply transports, who also have an initial position and a current autonomy level, among other parameters.

This scenario will be solved once the travel request of every customer has been served. For that, a Transport Agent must serve the request by picking the customer up at their initial location and dropping them off at their desired destination. At the same time, the movement of transports entails an expense of their autonomy. At some point, the Transport agents will need to recharge their batteries to keep serving customer requests. This will be done by driving to a charging station, start charging and wait until their autonomy is full again.

For the movement around the urban area, a routing service is used to calculate the fastest route between two points.

6.1.1. Transport agent's utility

To encourage Transport agents to solve the simulation scenario, we assigned to every customer travel request a benefit that will be reported upon completion. Besides the benefits, we defined also different costs that are associated with transport movement and charging fees. Our aim was to create costs and benefits derived from a realistic urban mobility scenario. For doing so, we defined the following values:

- **Benefits**: The benefits are modeled as monetary value obtained from serving customer agents. There is a STARTING_FARE price, which is a constant value added to the benefits once the customer is picked up. After that, the agent will obtain a fixed value PRICE_PER_KM every kilometer traveled during the service. These benefits modeling imitates the pricing model of a taxi service.

- **Costs**: There are two different costs defined. First, a PRICE_PER_KWh, which represents the price of charging one kilowatt in a charging station. Therefore, the cost derived from charging is proportional to the amount of kilowatts needed for a full charge. Second, we defined a travel PENALTY, that indicates what percentage of the transport agent's movement (in kilometers) must be penalized. This adds to the costs a numeric value equal to the penalty times the kilometers of any agent's movement. The PENALTY is usually a low value, 0.1 (10%) by default.

With this modeling, we achieve transport agents that are interested in serving customers with long trips, since they report more benefits, and, at the same time, picking up first the customers which are closer to them, since they entail less travel penalty. In this way, when planning, we are explicitly optimizing utility for the transport agents while implicitly reducing the waiting time of customers, since they will most likely be picked up by the transport which is closer to them.

$$Benefits = STARTING\_FARE + PRICE\_PER\_KM \cdot trip\_km \tag{6.1.1}$$

$$\begin{aligned} Costs &= TRAVEL\_COSTS + CHARGING\_COSTS \\ &= PENALTY \cdot travelled\_km + PRICE\_PER\_KWh \cdot KWh\_to\_charge \end{aligned} \tag{6.1.2}$$

6.1.2. Sources of conflict

The self-interest of the agents together with their utility function leads to a domain in which conflicts between agents easily arise. A conflict is a situation that invalidates the execution of the plans of two or more agents, which have actions that use the same resources and are thus incompatible inside the joint plan. A congestion, on the other hand, represents an interaction between the plans of two or more agent that causes the costs of the involved agent's plan to increase, causing an unpleasant situation which could be avoided modifying the agent's plan. During the BRPS process, these interactions are identified and dealt with, achieving a final feasible joint plan.

Customer conflicts

A customer conflict occurs when two or more agents aim to serve the same customer in their plans. In our simulations, once the travel request of a customer is served, the customer has completed its interaction with the system. Consequently, customers are only served once.

To solve this type of conflict, the planning process takes into account which transport agent is serving which customer at all times. A transport agent may plan to serve a customer which *has not been served yet* or a customer that *is being served already* by a different transport, *only if they pick up the customer before* their currently assigned transport. The factor that determines which agent will serve which customer is thus the pick up time, which is strongly related to the distance between the transport agent and the customer. This generally causes that customers are served by the transport which is closest to them, which is beneficial for them. The detailed process for conflict identification and management will be described in Section 6.4.

6.2 Actions

SimFleet's transport agents have two basic actions: attend a customer service, which involves moving to their location, picking them up and driving to their destination; and recharge their vehicle by driving to a charging station. All the different steps of each of the actions are encapsulated in SimFleet as a single atom. However, when it comes to planning, we must consider every step as an individual action with its own duration, even though the steps that belong to the same SimFleet action will always be executed consecutively. Besides that, taking into account the subsequent best-response algorithm, there is certain data relative to each of the steps of the whole action which is necessary for the process. For instance, the pick-up time of a customer, which is defined in the first step of a customer service, is used to check which agent is able to serve the customer before. Consequently, our system considers the following four actions:

1. PICK-UP: Move to a customer position and pick them up.
2. MOVE-TO-DEST: Move to a customer destination and drop them off.
3. MOVE-TO-STATION: Move to a charging station and wait for charge.
4. CHARGE: Begin charging until the vehicle is fully charged.

Actions 1 and 2 constitute a 'customer service' while actions 3 and 4 constitute a 'charging service'. During the construction of the individual plan of a transport agent, actions 2 or 4 will be only appear right after an action 1 or 3, respectively. In the same way, after actions 1 or 3 there can only appear actions 2 or 4.

6.2.1. Elements of an action

An action is defined in our system as a set of four elements: performing agent a, action type t, action attributes D and action statistics S.

$$\langle a, t, D, S \rangle$$

In this section we describe each element, focusing on the action's attributes and statistics.

- **agent (a)**: Name or ID of the agent performing such action.

- **type (t)**: Indicates the type of the action. There are three action types: CUSTOMER, CHARGE and RELOCATE.
- **attributes (D)**: Dictionary with different attributes of the action. The specific attributes of an action, which generally represent domain objects (agents, locations), are determined by the action type.
- **statistics (S)**: Dictionary with numerical data pertinent to the action. The statistics are also determined by the action type.

Attributes of an action

Action attributes represent its parameters; the set of objects to which the action makes reference. According to their type, actions have different attributes, which are explained below.

- PICK-UP action:
  - **customer_id**: ID of the customer agent that will be served in the action.
  - **customer_origin**: Origin position of the customer from which it will be picked up.
- MOVE-TO-DEST action:
  - **customer_id**: ID of the customer agent that will be served in the action.
  - **customer_destination**: Destination position of the customer in which it will be dropped off.
- MOVE-TO-STATION action:
  - **station_id**: ID of the station agent where the transport will charge.
  - **station_position**: Position of the station to which the transport will have to move.
- CHARGE action:
  - **station_id**: ID of the station agent where the transport will charge.

Statistics of an action

The statistics of an action include numerical data used for the evaluation of the plans and in the best-response procedure. In general, it has the time and distance values corresponding to the movement performed by the transport agent in such action. The statistics are different for each action type; for instance, in CHARGE actions, the charging time is also considered as a statistic. The specific statistics for each action type are presented below:

- PICK-UP action:
  - **time**: Time to move from the transport agent's position to the customer origin position.

  - **dist**: Distance traveled from the transport agent's position to the customer origin position.

- MOVE-TO-DEST action:
  - **time**: Time to move from the customer origin position to its destination.
  - **dist**: Distance traveled from the customer origin position to its destination.

- MOVE-TO-STATION action:
  - **time**: Time to move from the transport agent's position to the station's position.
  - **dist**: Distance traveled from the transport agent's position to the station's position.

- CHARGE action:
  - **time**: Time to fully charge the transport agent.
  - **need**: Amount of petrol/energy needed to fully charge the transport agent.

6.2.2. Action precalculation

With the information provided by the SimFleet configuration file, we are able to foresee the different actions that may take place during the execution or, from the planning perspective, that may be used in a transport agent's plan. To speed up the planning process, those actions are precalculated and stored in dictionaries.

To precalculate an action means to instantiate its elements with the adequate data, this process is equivalent to grounding as it is commonly named in planning contexts. Our precalculation process goes as follows: for each transport agent, we create:

- A PICK-UP and a MOVE-TO-DEST type action for every customer.
- A MOVE-TO-STATION and a CHARGE type action for every charging station.

These actions have values for their attributes but not for their statistics, since the latter are calculated according to the transport agent's position and/or autonomy at the time of executing the action. Consequently, the statistics of an action will be computed and assigned during the planning process.

In a similar way, we can also foresee which routes will the transport agents use during their execution. Transport agents can not plan to move freely within the urban area. Instead, every movement they do is determined by an action, whether is recharging or serving a customer travel request. Because of that, we consider only a finite set of positions, represented by pairs of coordinates, in which the transports and customers will be located. Those positions are the initial and destination positions of customer agents, initial positions of transport agents and charging station locations. Before the execution of the planner, the routes between each pair of points are requested to the routing service (explained in Chapter 2, subsection 2.1.3) and stored, saving again planning time.

Specifically, we precalculate:

- Routes from every *transport's initial position* to every *charging station location*, that will be used in case the first action of a transport agent is to recharge.
- Routes from every *transport's initial position* to every *customer's initial position*, that will be used in case the first action of a transport is to serve a customer request.
- Routes from every *charging station location* to every *customer's initial position*, that will be used in when a transport agent serves a customer request after charging.
- Routes from every *customer's initial position* to every *customer's destination position*, that will be used when a transport serves a customer travel request.
- Routes from every *customer's destination position* to every *customer's initial position*, that will be used when a transport serves a customer request right after finishing one.
- Routes from every *customer's destination position* to every *charging station locations* 's destination position, that will be used when the transport needs to recharge after serving a customer request.

By storing all these routes, we have covered every possible movement of transport agents in our domain.

6.3 Plans

A Plan is constituted by a list of actions ordered in ascending order according to their starting time. Every plan entry presents the corresponding action with its elements and initial and end time in seconds. An example of a plan in our domain can be seen in Table 6.1.

When it comes to our proposal, we must differentiate between two types of plans: *individual* and *joint* plans. Individual plans are the ones executed by a single agent; all its actions have the same agent as performer. A joint plan, however, is the union of every individual plan. The joint plan may contain conflicts among its actions, since the different plans that compose it where searched individually. For a joint plan to be executable, all the conflicts must be avoided.

6.4 Planning process

Our planner is designed to obtain feasible individual plans, taking into account both the simulation scenario and the current joint plan. The joint plan, as a union

init time	actions	end time
0,00	(Agent_A, PICK-UP, customer2)	4,62
0,00	(Agent_B, PICK-UP, customer1)	9,81
4,62	(Agent_A, MOVE-TO-DEST, customer2)	9,97
8,52	(Agent_C, PICK-UP, customer3)	17,51
9,81	(Agent_B, MOVE-TO-DEST, customer1)	16,07
17,51	(Agent_C, MOVE-TO-DEST, customer3)	25,41

Table 6.1: Visual representation of a plan. On the left column, the initial time instant of the action is presented in seconds. On the middle one, the action with all its elements. On the right column, the time instant in which the action finishes is indicated, also in seconds.

of individual plans, may present many conflicts. Such conflicts are taken into account by the Planner, which will avoid them when constructing a new individual plan. Therefore, every game round, agents will run the planner in their turn, obtaining their individual plan, which will be added to the joint plan. The best-response process will not finish while the joint plan presents any conflicts. Once the game converges, the joint plan is guaranteed to be conflictless and thus, executable.

In contrast to classical planning, described in Chapter 2.2.1, our modeling of the world state does not use literals, our actions are not exactly defined by preconditions and effects, and the goals of the plan are not explicitly defined nor known at the beginning of the process. Our domain is very limited, and the planning process is completely guided by the transport agent's self-interest. Transport agents aspire to obtain as much benefit from its execution as possible. This is equivalent to search for the plan with highest utility value. Since the benefits of the plan are given by completing customer actions, to achieve the greatest utility value the planner considers every *reachable* customer agent as an open goal. A customer is reachable if 1) no other transport agent is picking the customer up before the current agent and 2) the current agent has enough autonomy to complete the customer service.

A transport agent is interested in serving as many customers as possible. Consequently, the main type of action the planning process will generally consider is serving customer requests; picking a customer up and driving them to their destination. Only when the autonomy of the agent is not full, the actions that constitute a charging service will be considered.

The current world state is then represented by the transport agent's current position and autonomy as well as the current reachable open goals. These two are the only elements which matter to the transport agent (and therefore to the planning process). The attributes regarding other agents in the scenario are not taken into account; the customers that have been already served and are not reachable are ignored by the planning transport agent. When actions are applied, the state is modified by updating the transport agent's location, autonomy and/or the list of open goals.

In contrast to the approach of [25], our planner does not look for a better plan[1] but for the best one, thus performing optimal planning, which is reasonable because of how restricted our domain is. The individual plan returned from the planning process must be a best response to the individual plan of every other transport agent. Consequently, when planning, we are not just looking for a feasible plan, but for the best possible plan given the current state of the scenario.

As for the plan-searching process, the planner obtains the optimal plan by building a search tree whose nodes are partial plans. The open nodes (nodes that have not been expanded yet) are evaluated and stored in a priority queue, so that nodes with better value are explored first. Expanded nodes that generate no children are considered solutions, and the plans they contain as complete plans. Solution nodes are evaluated as such and stored in a list. The search finishes when the queue of open nodes gets emptied. Then, the best solution found is returned.

In this section we describe, in depth, the elements of our planner and the components of its search tree, as well as the procedure used to build and explore it.

6.4.1. Planner elements

Every instance of our planner has certain elements which are necessary for the plan-searching process. These elements are passed by the BRPS procedure (Chapter 7) upon creation of the planner instance and may be consulted and/or updated during the planning process. Following, we introduce each of those elements, describing their use.

Firstly, the planner has access to the **dictionary of precalculated actions and routes** (Section 6.2.2). When building the search tree, the precalculated actions are checked, and the tree is expanded creating new ramifications that include the feasible ones. On the other hand, once an action is selected to include it in a plan, its statistics (Section 6.2.1) must be filled, which is done by using the transport agent's current location and autonomy. All actions include a statistic that indicates the time it takes to be completed. For the actions that indicate a movement, this time is calculated as the traveled distance divided by the transport agent's speed. Such distance is given by the route, precalculated in the aforementioned dictionary.

The planner includes also a dictionary which stores the **transport agent's attributes**, extracted from the simulation configuration file during the initialization of the planner. This includes the agent's ID, its initial position, initial autonomy, maximum autonomy, and speed. The initial position and autonomy are used for the creation of the first planning search-tree nodes. The maximum autonomy indicates the autonomy the transport agent will have after a charging service. Finally, the speed is used to calculate the time in any action that includes a movement.

[1]Better plan refers to a plan which reports higher utility than the plan the agent had found on the previous iteration of the BRPS process.

Finally, each planning instance has its own **Table of Goals**, which follows the structure presented in Table 6.2. The Table of Goals keeps updated information about the current reachability of customers in the scenario. It indicates, for every customer, if a transport agent is serving its travel request. If so, it will show the serving transport agent's ID and the customer pick-up time. The Table of Goals is consulted every time a customer service action is considered for adding to a search-tree node. If the current transport agent can not pick the customer up before the time indicated in the Table of Goals, such customer is considered non reachable and the action to pick it up is therefore discarded. The Table of Goals is then used by every transport agent to determine the open goals of their individual plan.

goal	agent	pick-up time
customer 1	Agent C	2,26
customer 2	Agent A	4,62
customer 3	Agent B	3,95

Table 6.2: Visual representation of a Table of Goals. Each row presents a customer, its serving agent and pick-up time, in seconds

When an agent proposes a plan, the joint plan is updated and a new Table of Goals is computed. Every entry from the joint plan that contains a PICK-UP action is stored, grouping them by served customer ID. Then, the action with earliest pick-up time for every customer is chosen. The customer gets assigned the transport agent and pick-up time that such action indicates.

6.4.2. Plan evaluation

The value of a plan is tied to the utility it reports to the transport agent that executes it. Only individual plans are evaluated, since joint plans are not better or worse, they are simple feasible or non feasible. A plan can be evaluated in three different situations:

1. As a partial plan in a search tree node.
2. As a complete plan in a leaf (solution) node.
3. As a part of a joint plan.

This subsection discusses plan evaluation in the second and third cases, whereas the first is explained in Section 6.4.3, where the evaluation of tree nodes (partial plans) is explained in detail.

During the planning process, every expanded node which generated no children (leaf node) is considered as a complete plan or solution. The evaluation of a complete plan is performed following the equations in Section 6.1.1. The plan reports benefits for every customer service, which are reduced by the costs of charging the vehicle and the traveling penalty. In this case, the evaluation of the plan is equal to the utility that the agent gains by executing it. When the queue of open nodes is exhausted, the plan search process finishes and the complete plan with highest utility is returned to the BRPS process (Chapter 7), which integrates it in the joint plan.

The utility of plan π will remain unchanged as long as the customers its performing agent is serving are not served by any other agent. Since the returned plans are in best response, π reports the maximum utility its performing agent can achieve at the moment of its planning. Nevertheless, as soon as another agent proposes another plan, π may have stopped being in best response, having its utility reduced.

When a plan is evaluated as part of a joint plan, it may present conflicts. If the plan does not bring any conflicts, its utility will be the same it had when returned. However, when a plan has any conflict, its cost is usually increased, which encourages its performing agent to avoid conflicts. A conflict arises when an agent which planned to serve a certain customer gets its goal stolen by another participant agent, which plans after it, and serves the same customer earlier (Section 6.1.2). In our design, when a plan presents customer conflicts, the benefits that would be reported by serving the customer in conflict are not considered; only the costs are. This reduces drastically the utility of plans in conflict, since customers are the only source of benefits. Therefore, it stimulates our transport agents to look for other customers to attend in the subsequent planning instances of the next iterations of the BRPS process.

6.4.3. Partial plan search tree

Our planner searches for the optimal plan by building and expanding a search-tree of partial plans. The nodes of the tree contain partial plans. Nodes expand and generate children, which inherit their plan and add new actions. Therefore, partial plans are built incrementally by adding customer or charging services to them, one step at a time.

Nodes

A node of the search tree represents a partial plan. For that, each node presents a list with the actions that its partial plan contains. In addition, it also contains the transport agent's position and autonomy at the end of the execution of the partial plan, together with the time instant in which the last action of the partial plan is completed (node end time). Finally, nodes contain also a list of completed goals. In case their partial plan includes actions that serve a customer request, such customer will be indicated as a completed goal.

This representation is not the common one in classical planning or Partial Order Planning, in which each node represents an action. However, our representation of partial plans (that include several actions) in nodes of a tree allows us to have a reduced branching factor when compared with other planning representations. This is possible in our case because our planner is ad-hoc to the domain, and hence, we can take profit of the characteristics and restrictions of the domain.

As with any kind of tree, nodes have one connection to their parent node and one connection per children node, if any. Every time an open node is expanded its children are spawned. Those children inherit the partial plan defined by the parent and update it by adding new actions. If a node is expanded but returns no children, it is considered a leaf node or solution and its partial plan a complete plan.

Nodes can be of two types: customer or charge nodes. A customer node is one that adds to its parent's partial plan the necessary actions to serve a customer request. In contrast, a charge node adds the actions of moving to a station and charging, which constitutes a charging service.

When a node is expanded, the planner creates all its possible customer-node children. If during this process one (or more) of the customer-node children is discarded because the agent's autonomy does not allow it to complete its customer service, the process returns a flag which indicates that charge-node children must be created too. Therefore, charge-node children will only be created if at least one customer can not be served because of the agent's low autonomy. The planner spawns one charge-node children per charging station in the scenario. As it can be deduced from above, the branching factor of the search tree will be mostly affected by the amount of customers and charging stations in the simulation configuration.

After their creation, nodes are evaluated by an A*-like evaluation function. The value of a node is the sum of the utility reported by its current partial plan and an heuristic estimate of the utility that could be achieved once the plan is complete. Leaf nodes, however, are evaluated as complete plans, following the procedure described in Section 6.4.2. The value of each node is used to define expansion priorities during the plan search process.

Customer nodes creation

The creation of customer nodes is encapsulated in a function which can receive a parent node selected to expand or nothing. In the latter case, a new node will be created from scratch. In general, the function creates one children node per reachable customer, serving their travel request. As we mentioned above, a customer is reachable if (1) no other transport agent is picking the customer up before the current agent and (2) the current agent has enough autonomy to complete the customer's service. To check these conditions, the parent node's end time is taken into account as well as its list of completed goals and the Table of Goals. After checking (1), the autonomy consumption of the customer trip is computed and compared towards the parent node's current autonomy. If (2) does not hold, the function will return a flag, indicating that at least one customer was unreachable because of not having enough autonomy.

Analyzing at plan level, each children node updates the partial plan of its parent by adding a PICK-UP and a MOVE-TO-DEST action that make reference to a unique, reachable customer.

Charge nodes creation

Charge nodes will be created whenever the autonomy of the agent is not maxed out in a parent node. With this, we avoid ramifications of the search tree where charging actions are applied senselessly, thus reducing the tree's branching factor. These tree branches would eventually be discarded naturally by the planning algorithm, since the action of charging entails no benefits, only costs. Therefore, as the utility is optimized, the amount of charging actions is reduced.

The creation of charging nodes is also encapsulated in a function. In general, it creates one children per charging station in the simulation scenario, updating its parent's partial plan with the corresponding MOVE-TO-STATION and CHARGE actions.

Node evaluation

When building the search three, the order in which nodes are expanded can greatly affect the time and memory consumption of the process. Aiming to efficiently manage the computational power needs of our planner, we decided to use an A* algorithm to decide which node to expand at every moment. Every generated node is evaluated and enqueued in a heap of open nodes. Their priority in the queue is directly proportional to their evaluation score; i.e: their f-value. To evaluate the nodes we make use of the classical equation, presented in Equation 6.4.1, where g is the utility of the partial plan the node represents and h is an optimistic calculus of the expected benefits that completing every reachable non-completed goal would yield.

$$f(x) = g(h) + h(x), \;\; h(x) \leq h^*(x) \; \forall \, x \tag{6.4.1}$$

The heuristic function is a relaxation of the problem constraints. It neglects the time taken to pick up the reachable customers and dismisses the costs that entail serving their requests, considering only the benefit that their trip would return. The h value of a node is therefore the addition of the benefits derived from serving all customers that have not been served by the node's end time. In other words, the transport agent assumes that it can serve every available customer by itself. This heuristic is substantially optimistic, since even in the best case scenario the costs would be added to the real utility.

Search space exploration algorithm

The planning process begins by creating the initial nodes. According to the Table of Goals (extracted from the previous round's joint plan) and the own agent's initial autonomy, the planner will create:

1. One customer node per reachable customer.

2. One charge node per charging station *iff* the autonomy of the transport agent is not at its maximum value.

These nodes will be evaluated and added to a priority queue that stores open nodes giving priority to the ones with higher value.

Following, the algorithm's main loop begins. While there are open nodes in the queue, a node is selected for expanding. Its customer-type children are created and, if necessary, its charge-type children are as well. Every child is evaluated and stored in the queue only if its value is higher than the value of the best solution found so far (if any). Meanwhile, the parent node gets discarded.

If the children creation processes return no children, the expanded node is a solution node and it is therefore reevaluated as a solution (without heuristic value) and stored in the list of solution nodes. The value of the solution is compared against the best solution found, which will be replaced if it is surpassed.

Once the open node queue becomes empty, the plan in the best solution node is extracted and returned as solution.

By creating customer and charge-type children, we are building different plans by adding, in each step, two actions to the previous partial plan. These two actions can be either a PICK-UP, MOVE-TO-DEST couple, completing a customer request or a MOVE-TO-STATION, CHARGE couple, completing a recharge of the transport agent's autonomy. We decided to build plans in this way because of two reasons.

First, even though our domain has 4 actions, those can not be ordered freely inside a plan. After picking a customer up, a transport agent can only drive them to their destination. It can not move to a station or charge while the customer is boarded. In a similar way, if the transport agent needs to charge and moves to a station, its next action must be to charge. This causes that actions appear always in the aforementioned couples.

Second, by using this method we only consider the addition of necessary and feasible actions every time. Consequently, we are avoiding tree ramifications that would eventually be discarded either because of conflicts or because of a low utility value.

These two features of our domain drastically reduce the branching factor of the search tree. A visual representation of a tree branch can be seen in Figure 6.1, where the construction of a complete plan from an empty plan is presented.

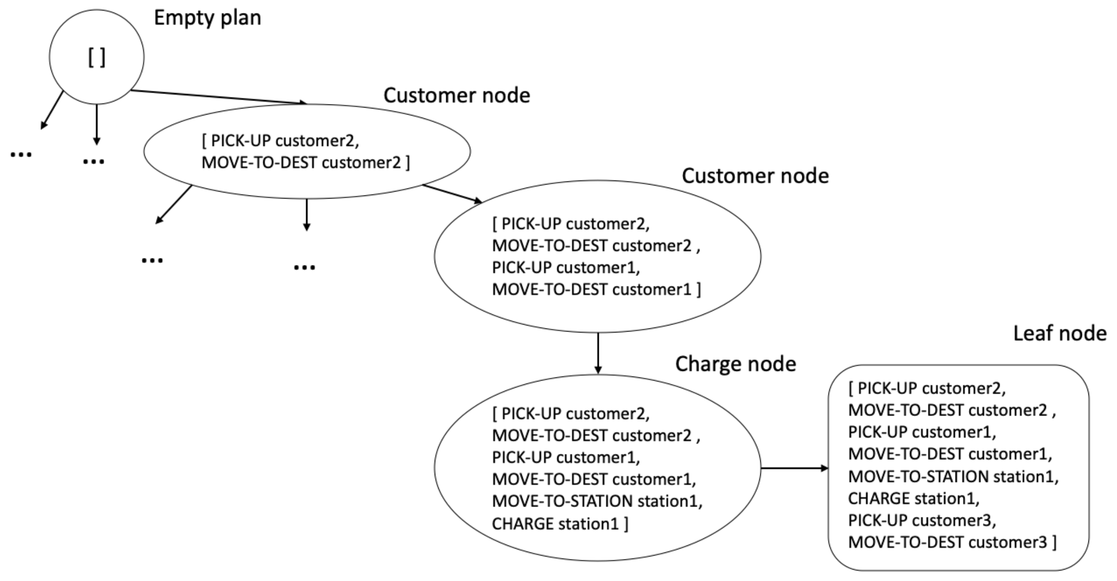

Empty plan
[ ]
Customer node
[PICK-UP customer2, MOVE-TO-DEST customer2]
Customer node
[PICK-UP customer2, MOVE-TO-DEST customer2 , PICK-UP customer1, MOVE-TO-DEST customer1]
Charge node
[PICK-UP customer2, MOVE-TO-DEST customer2 , PICK-UP customer1, MOVE-TO-DEST customer1, MOVE-TO-STATION station1, CHARGE station1]
Leaf node
[PICK-UP customer2, MOVE-TO-DEST customer2 , PICK-UP customer1, MOVE-TO-DEST customer1, MOVE-TO-STATION station1, CHARGE station1, PICK-UP customer3, MOVE-TO-DEST customer3]
...

Figure 6.1: Partial-plan tree branch. Rounded shapes represent nodes with partial plans, which can be of customer or charge type, according to the actions they add to their parent's plan. Arrows represent a kinship relation. The leaf node (lower right corner) presents a complete plan.

6.5 Search tree pruning

Planning is a computationally hard task. Our planner reduces the computational cost by building plans exclusively for our domain, taking advantage of its characteristics to avoid useless plans which would end up discarded, and it also reduces computation due to the representation of partial plans in the nodes of the search tree instead of the classical representation of actions. Nevertheless, we included some features which aid to speed up the plan search process as well as lower memory consumption. In general, during the generation of the search tree, there are nodes which can be safely discarded, and thus never expanded, since its properties indicate that either it will not build a valid plan or the plan it produces will not be better than the best plan found so far.

6.5.1. Best Solution prune

Once the first leaf node is found, its plan is extracted, evaluated and its utility saved as the *best solution value* found so far.

Whenever the process finds another solution, the utility of its plan will be compared to the best solution value, updating the latter if it overcomes it.

Each time a node is extracted from the priority queue to be expanded, if its f-value is lower than the best solution value, it is instead discarded. This can be done safely because the f-value of an open node is the sum of the node's partial plan utility and the best expected utility that a plan deriving from that node could achieve. Such expected utility is returned by an optimistic heuristic function (Section 6.4.3). The partial plan of an open node with an f-value below the best solution value has no potential to evolve into a better solution, and so the planner can avoid wasting time expanding it.

6.5.2. Storage of Partial Solutions

All leaf nodes in our search tree are customer-type nodes. This is because it would make no sense for the agents to plan to charge if, after charging, they can not serve any customer, since charging entails costs. Because of that, every generated customer node can be a potential solution.

In general, solutions or complete plans are detected when, upon expanding a node, it returns no children. However, we decided to store *partial solutions*. A partial solution is an open customer node which is evaluated as a solution and saved in the list of solutions. Therefore, every spawned customer-node children is both saved as a partial plan in the queue of open nodes and as a partial solution in the list of solutions.

In this way, our planning algorithm has available solutions to perform the best solution prune (Section 6.5.1) almost from the beginning of the planning process, avoiding the need to detect a complete plan to start pruning. Since partial solutions are partial plans evaluated as complete plans (without heuristic value), their utility will always be lower than the utility obtained by a complete plan, which makes it safe to store them as solutions.

6.5.3. Previous Plan Utility bound

When an instance of the planner is created, a previous plan of the invoking agent can be passed. If there is a previous plan and the utility it reports is higher than 0, the utility will be used to define a lower bound value for the planning process. When a node is evaluated, if its value is below the lower bound, it will be discarded. In this way, leaf nodes which contain solutions that are worse than the previously obtained one are not considered, speeding the process up considerably.

6.6 Planning in large scenarios

The complexity of our planning scenarios is proportional to the number of customers and charging stations that it includes. Since node expansion generally creates one customer-child node per reachable customer and one charge-child node per station, it can easily be deduced that great amounts of them will increase the search tree branches.

For small and medium domains, this does not suppose a problem, since we have many measures to save computational power during our planning process. However, we found that for big problems (20+ nodes per expansion) some method was needed for obtaining initial plans.

Initial plans are created at the start of the BRPS process when there is no previous plan the utility of which can be used as lower bound for the search. Besides that, the agent which plans in the first place does not have information on the Table of Goals that limits its possibilities, thus facing a large number of feasible plans. Once the agents have a previous plan and the Table of Goals contains information, the planning process is drastically speed up, even for big scenarios.

Consequently, to palliate this issue we developed two independent methods which can be applied in the planner instance of each agent during the first iteration of the BRPS to reduce the time needed to propose the initial plans.

6.6.1. Goal Limitation

A way in which we can avoid too prolonged planning processes is to limit the amount of open goals that the agent aims to complete. In general, an agent will plan to achieve as many goals as possible; in other words, to serve as many customers as possible. We implemented a method that limits the goals to a percentage of the total. Every time a customer node is spawned, the number of goals in its list of completed goals is checked. If it has served such percentage of customers, the node is considered a solution and thus not included in the queue of open nodes.

This limitation makes sense since no transport agent would be able to serve all customers in any scenario considering there are other transport agents competing for those customers.

6.6.2. Initial Feasible Joint Plan

The planning during the first iteration of the BRPS process is considerable slower because of the absence of previous individual plans. For that, we implemented a method that creates initial plans for every agent, making sure there are no conflicts among them.

The creation of the initial plan is performed as follows: First, the amount of customers is divided by the amount of transport agents, obtaining the number of goals (x) each agent will initially complete. Then, x different customers are assigned to each agent randomly. If the division is not exact, some transports will have $x+1$ or $x-1$ customers assigned instead. Then, a greedy algorithm is used by each agent to plan how to serve their customers. The algorithm makes sure that the agent serves the closest customer at every time, charging in the closest station whenever necessary. Therefore, in every iteration of the greedy planning, the process checks if the agent has enough autonomy to pick up its currently closest customer. If so, the agent will pick it up and repeat this. If not, the agent will charge in its closest charging station, then pick up the closest customer. Once the x customers are served, the process returns the plan.

With this, we obtain a feasible joint plan which is used to fill the Table of Goals, providing the planner with the necessary tools to speed up the search.

The goal limitation and the creation of an initial, feasible joint plan have proved to be very effective, reducing the amount of generated nodes during the initial planning process greatly. However, they have an influence on the BRPS process, as they guide it towards certain equilibria, avoiding others which can not be reached with the limitations the methods impose.

6.7 Plan building example

In this section we illustrate the plan building process by showing how a complete plan is developed in a small scenario, presented in Figure 6.2, which contains three customers and one charging station. In this case, we will show the construction of the initial plan for Agent A, who proposes its plan before any other agent. From the planning perspective, this means that the Table of Goals is completely empty and there is no previous plan whose utility defines a lower bound.

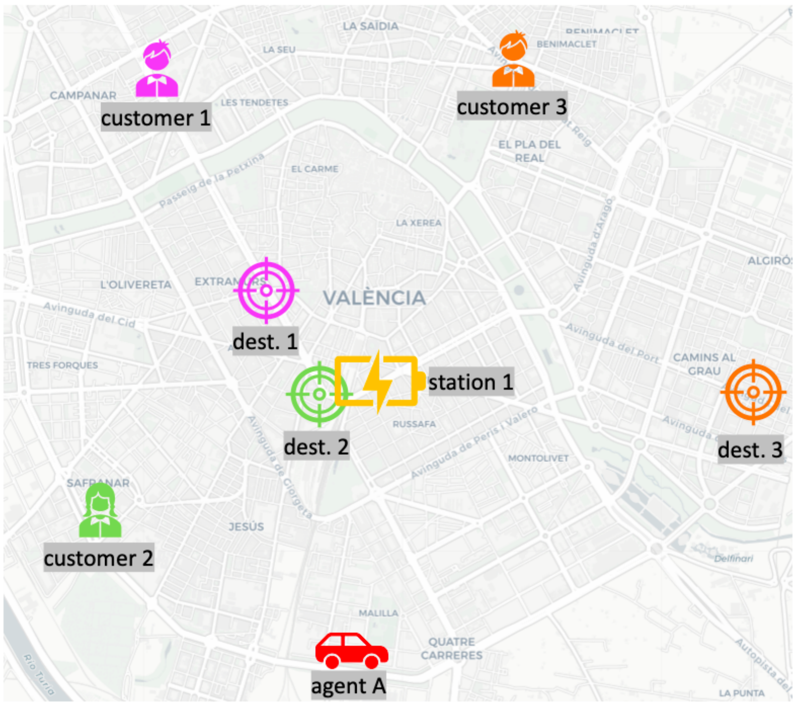


Figure 6.2: Small planning scenario based on the city of Valencia, Spain

Hereunder we show many graphs, each representing an iteration of the search process. Take into account that the nodes present written in them only the instructions they add to their parent's partial plan. Therefore, to know the partial plan that a node represents, one only has to join the instructions of every previous node related to it. Also, for the sake of simplicity, we have considered that charge nodes are only generated whenever the agent can not reach a customer because of not having enough autonomy. In contrast with this, as explained in Section 6.4.3, charge nodes are generally created when the agent's autonomy is not full. To understand every element of the search tree, please refer to the legend in Table 6.3.

The process begins with one empty node, which acts as the tree root. It gets expanded, generating one children node per reachable customer which, in this case, makes a total of three nodes (see Figure 6.3). The numbers in the lower

◯	Partial plan node
▢	Complete plan node
⭕	Expanding node
🟩	Returned solution node
→	Parent - child relation
✕	Expansion with no children

Table 6.3: Legend for the plan building example

right corner indicate the f-value of each node. A higher value implies a higher expansion priority.

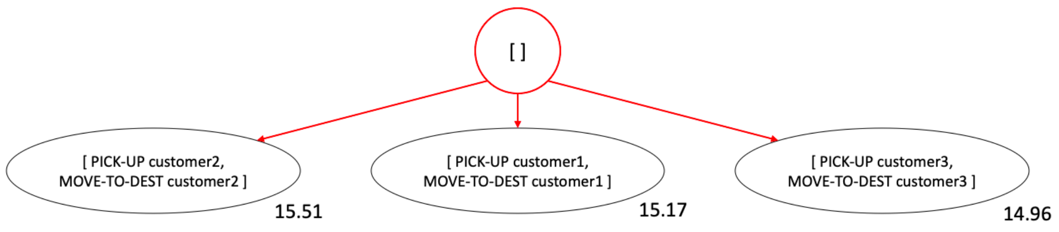

[ ]
[PICK-UP customer2, MOVE-TO-DEST customer2]
15.51
[PICK-UP customer1, MOVE-TO-DEST customer1]
15.17
[PICK-UP customer3, MOVE-TO-DEST customer3]
14.96

Figure 6.3: Initialization

Following the order established by their priorities, the nodes are expanded. In Figure 6.4, the expanded node generates two customer nodes, which indicates that the agent has enough autonomy to serve any customer after serving customer 2. However, in Figures 6.5 and 6.6, the expanded nodes generate a single charge node, since the agent is not able to serve any customer after serving customer 1 or 3 without charging first.

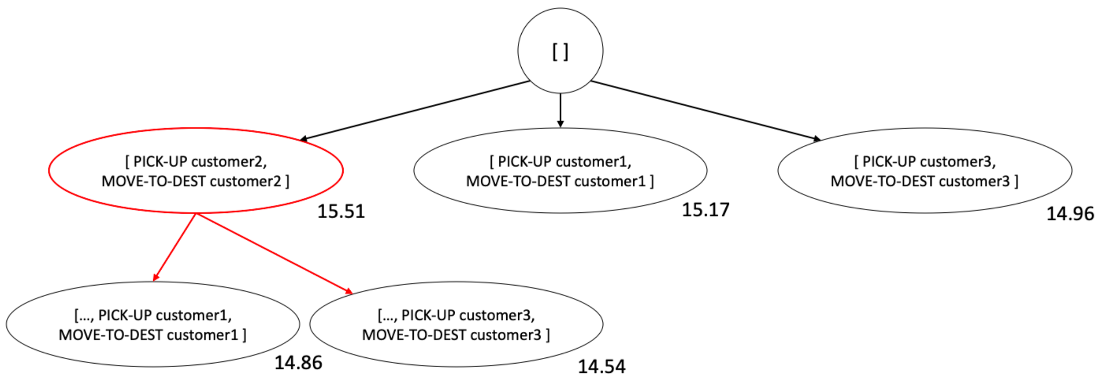

[ ]
[PICK-UP customer2, MOVE-TO-DEST customer2]
15.51
[PICK-UP customer1, MOVE-TO-DEST customer1]
15.17
[PICK-UP customer3, MOVE-TO-DEST customer3]
14.96
[..., PICK-UP customer1, MOVE-TO-DEST customer1]
14.86
[..., PICK-UP customer3, MOVE-TO-DEST customer3]
14.54

Figure 6.4: Iteration 1

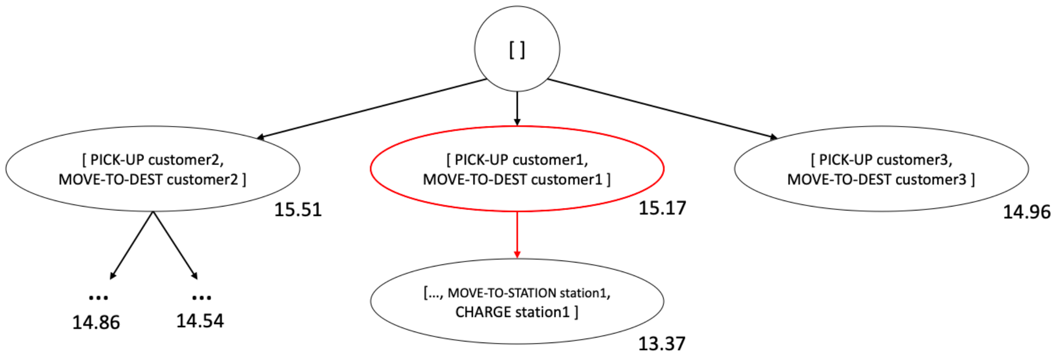

[ ]
[PICK-UP customer2, MOVE-TO-DEST customer2]
15.51
[PICK-UP customer1, MOVE-TO-DEST customer1]
15.17
[PICK-UP customer3, MOVE-TO-DEST customer3]
14.96
...
14.86
...
14.54
[..., MOVE-TO-STATION station1, CHARGE station1]
13.37

Figure 6.5: Iteration 2

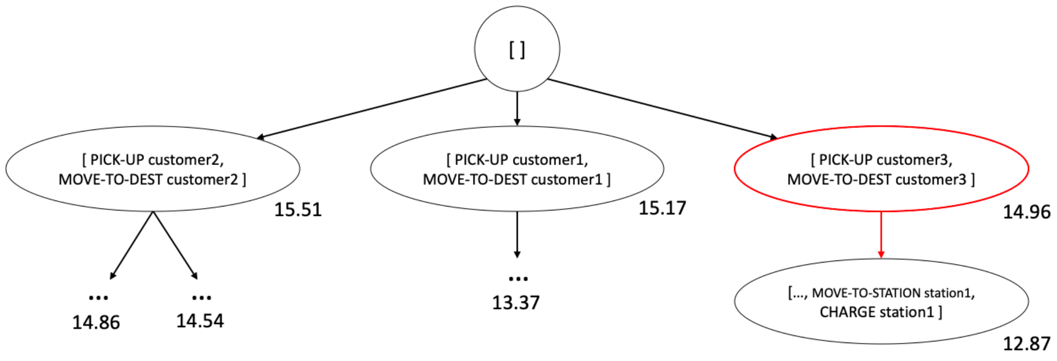

[ ]
[PICK-UP customer2, MOVE-TO-DEST customer2]
15.51
[PICK-UP customer1, MOVE-TO-DEST customer1]
15.17
[PICK-UP customer3, MOVE-TO-DEST customer3]
14.96
...
14.86
...
14.54
...
13.37
[..., MOVE-TO-STATION station1, CHARGE station1]
12.87

Figure 6.6: Iteration 3

As it can be seen in Figures 6.7 and 6.8, after serving two customers, the agent is obligated to charge to serve the third one. We can already intuit that the left-most branch of the tree will contain the best solution, since through its plans, the agent manages to serve two customers without the need of charging, which indicates a better customer serving order.

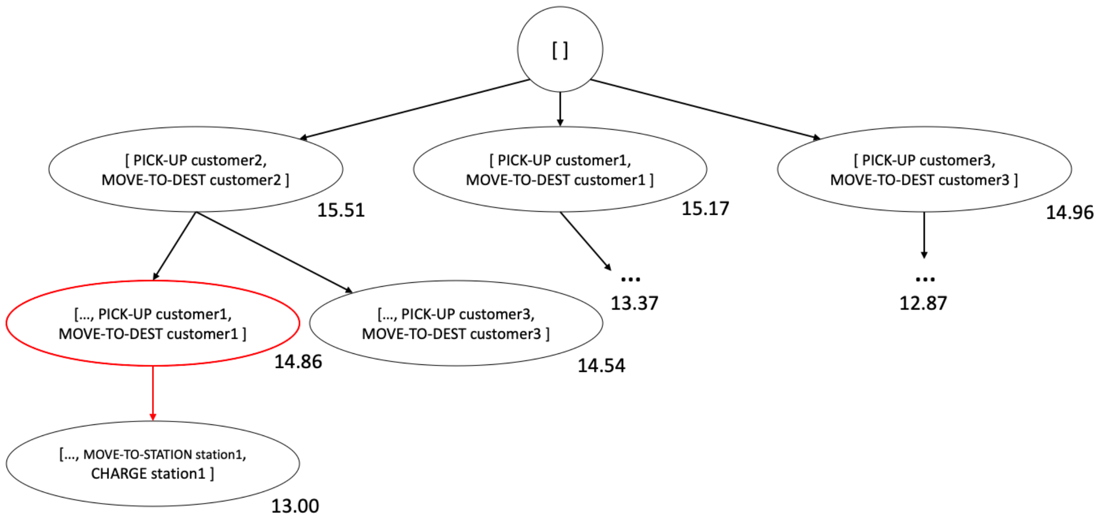

[ ]
[PICK-UP customer2, MOVE-TO-DEST customer2]
15.51
[PICK-UP customer1, MOVE-TO-DEST customer1]
15.17
[PICK-UP customer3, MOVE-TO-DEST customer3]
14.96
[..., PICK-UP customer1, MOVE-TO-DEST customer1]
14.86
[..., PICK-UP customer3, MOVE-TO-DEST customer3]
14.54
...
13.37
...
12.87
[..., MOVE-TO-STATION station1, CHARGE station1]
13.00

Figure 6.7: Iteration 4

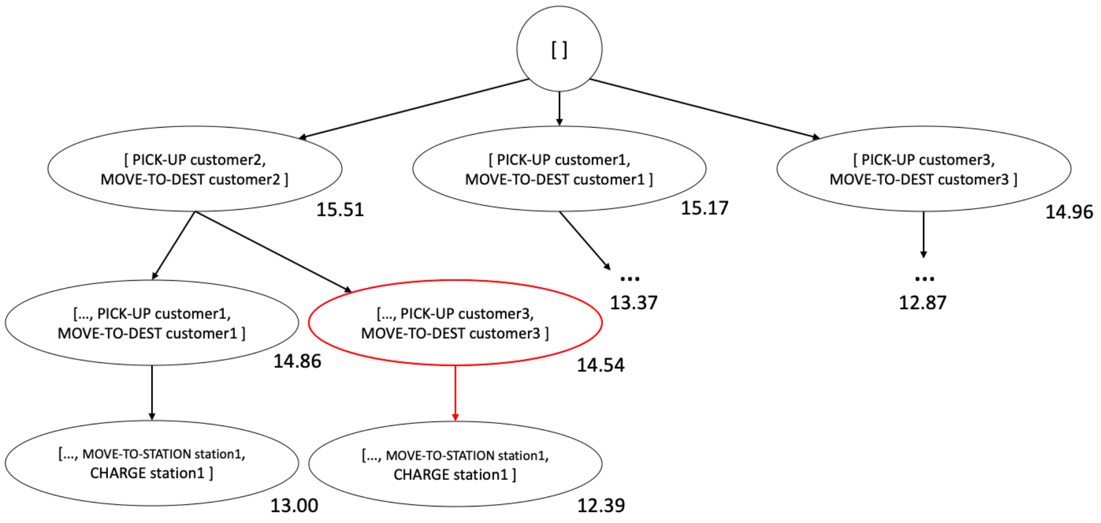

[ ]
[PICK-UP customer2, MOVE-TO-DEST customer2]
15.51
[PICK-UP customer1, MOVE-TO-DEST customer1]
15.17
[PICK-UP customer3, MOVE-TO-DEST customer3]
14.96
[..., PICK-UP customer1, MOVE-TO-DEST customer1]
14.86
[..., PICK-UP customer3, MOVE-TO-DEST customer3]
14.54
...
13.37
...
12.87
[..., MOVE-TO-STATION station1, CHARGE station1]
13.00
[..., MOVE-TO-STATION station1, CHARGE station1]
12.39

Figure 6.8: Iteration 5

Following, in Figure 6.9 we can see how in the middle branch, after charging, the agent is able to pick up any of the remaining customers.

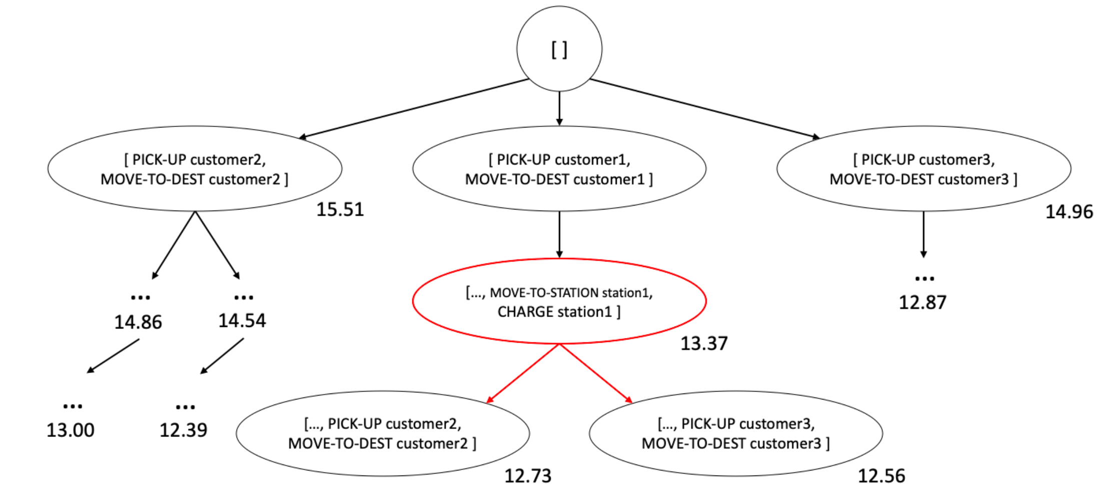

[ ]
[PICK-UP customer2, MOVE-TO-DEST customer2]
15.51
[PICK-UP customer1, MOVE-TO-DEST customer1]
[PICK-UP customer3, MOVE-TO-DEST customer3]
14.96
...
14.86
...
14.54
[..., MOVE-TO-STATION station1, CHARGE station1]
13.37
...
12.87
...
13.00
...
12.39
[..., PICK-UP customer2, MOVE-TO-DEST customer2]
12.73
[..., PICK-UP customer3, MOVE-TO-DEST customer3]
12.56

Figure 6.9: Iteration 6

Then, Figure 6.10 shows how the leftmost branch expands with a single customer node that constitutes the first complete plan of the tree, since there are no more customers to serve. Nevertheless, the node is stored as an open node; it will be identified as a solution once the algorithm tries to expand it. It is important to mention that the pruning techniques (Section 6.5) have been active since the start of the process and therefore, thanks to the storage of partial solutions (Section 6.5.2), every customer node that has been generated was also evaluated as a complete solution and stored as such. This is important because, even though the lastly generated node is not a considered solution yet, a copy of it has been stored as such, with an utility value of 12.19. Because of that, from now on, once a node is expanded, any children with f-value lower than 12.19 will be discarded.

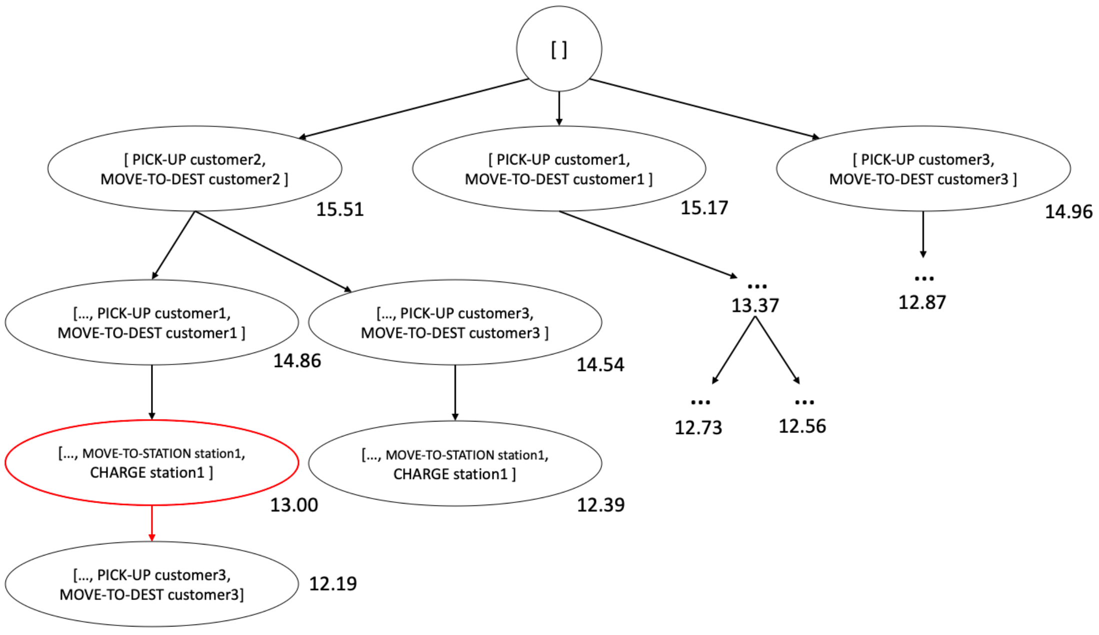

[ ]
[PICK-UP customer2, MOVE-TO-DEST customer2]
15.51
[PICK-UP customer1, MOVE-TO-DEST customer1]
15.17
[PICK-UP customer3, MOVE-TO-DEST customer3]
14.96
[..., PICK-UP customer1, MOVE-TO-DEST customer1]
14.86
[..., PICK-UP customer3, MOVE-TO-DEST customer3]
14.54
...
13.37
...
12.87
...
12.73
...
12.56
[..., MOVE-TO-STATION station1, CHARGE station1]
13.00
[..., MOVE-TO-STATION station1, CHARGE station1]
12.39
[..., PICK-UP customer3, MOVE-TO-DEST customer3]
12.19

Figure 6.10: Iteration 7

In Figure 6.11, we can see the effect of the aforementioned pruning method. Although the agent has just charged, it has generated a single child only picking up one customer. We have avoided the generation of a customer node with no potential to overpass the already found partial plans.

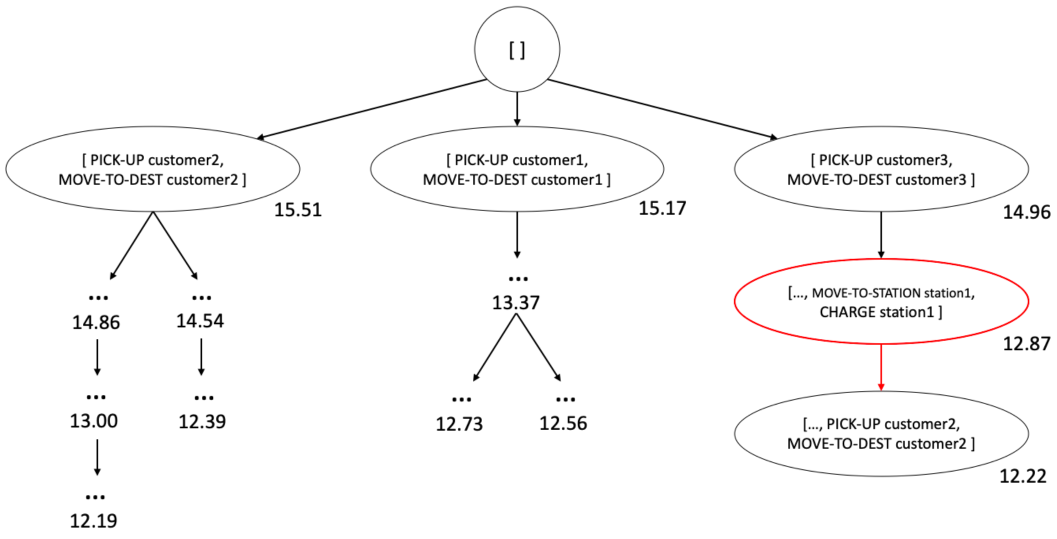

[ ]
[PICK-UP customer2, MOVE-TO-DEST customer2]
15.51
[PICK-UP customer1, MOVE-TO-DEST customer1]
15.17
[PICK-UP customer3, MOVE-TO-DEST customer3]
14.96
...
14.86
...
14.54
...
13.37
[..., MOVE-TO-STATION station1, CHARGE station1]
12.87
...
13.00
...
12.39
...
12.73
...
12.56
[..., PICK-UP customer2, MOVE-TO-DEST customer2]
12.22
...
12.19

Figure 6.11: Iteration 8

In the following iterations, presented in Figures 6.12, 6.13, 6.14 and 6.15, it is shown how none of the expanded nodes generate any children, therefore turning into leaf nodes. Even though their partial plans still have open goals (at least one customer to pick up), their children were discarded because none of them could surpass the 12.19 utility of the previously found partial plan.

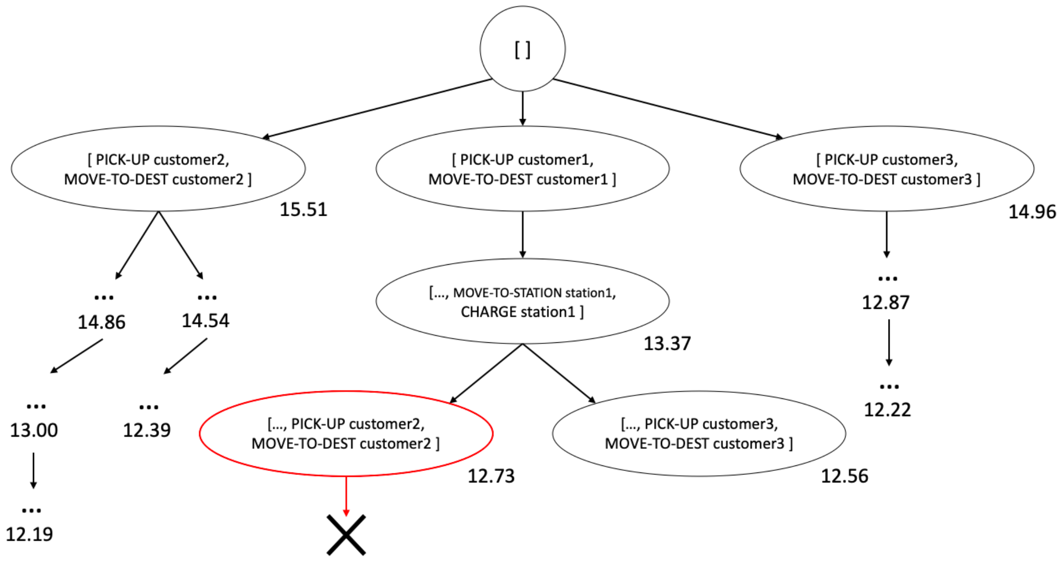

[ ]
[PICK-UP customer2, MOVE-TO-DEST customer2]
15.51
[PICK-UP customer1, MOVE-TO-DEST customer1]
[PICK-UP customer3, MOVE-TO-DEST customer3]
14.96
...
14.86
...
14.54
[..., MOVE-TO-STATION station1, CHARGE station1]
13.37
...
12.87
...
13.00
...
12.39
[..., PICK-UP customer2, MOVE-TO-DEST customer2]
12.73
[..., PICK-UP customer3, MOVE-TO-DEST customer3]
12.56
...
12.22
...
12.19

Figure 6.12: Iteration 9

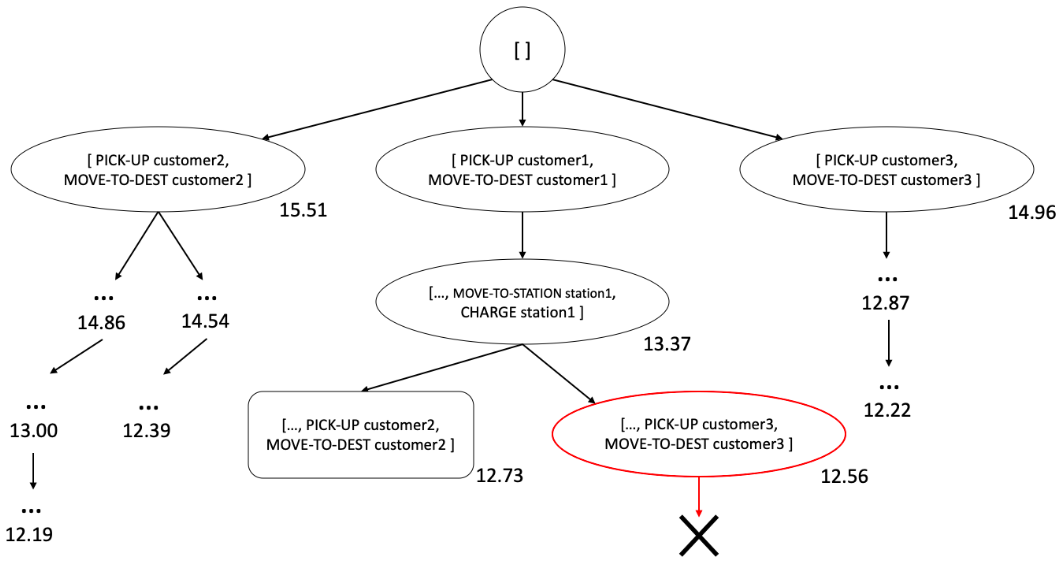

[ ]
[PICK-UP customer2, MOVE-TO-DEST customer2]
15.51
[PICK-UP customer1, MOVE-TO-DEST customer1]
[PICK-UP customer3, MOVE-TO-DEST customer3]
14.96
...
14.86
...
14.54
[..., MOVE-TO-STATION station1, CHARGE station1]
13.37
...
12.87
...
13.00
...
12.39
[..., PICK-UP customer2, MOVE-TO-DEST customer2]
12.73
[..., PICK-UP customer3, MOVE-TO-DEST customer3]
12.56
...
12.22
...
12.19

Figure 6.13: Iteration 10

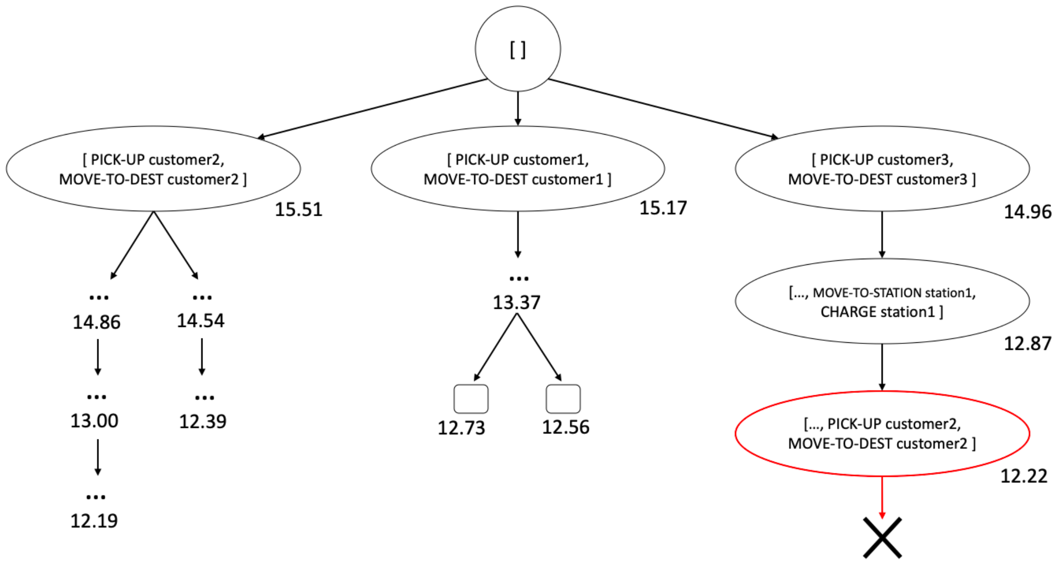

[ ]
[PICK-UP customer2, MOVE-TO-DEST customer2]
15.51
[PICK-UP customer1, MOVE-TO-DEST customer1]
15.17
[PICK-UP customer3, MOVE-TO-DEST customer3]
14.96
...
14.86
...
13.00
...
12.19
...
14.54
...
12.39
...
13.37
12.73
12.56
[..., MOVE-TO-STATION station1, CHARGE station1]
12.87
[..., PICK-UP customer2, MOVE-TO-DEST customer2]
12.22

Figure 6.14: Iteration 11

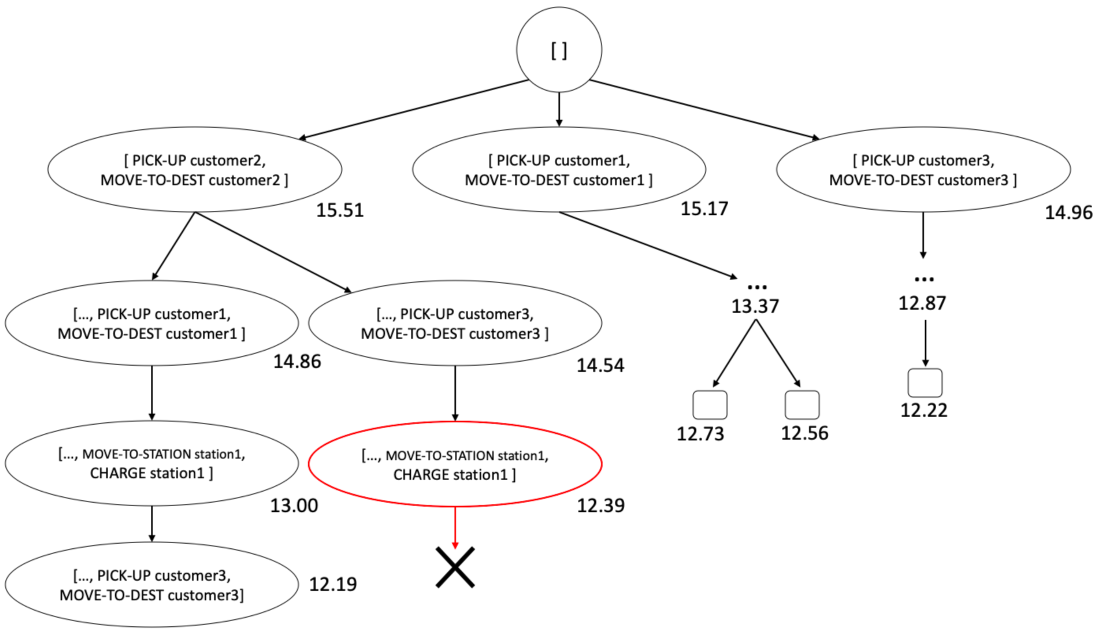

[ ]
[PICK-UP customer2, MOVE-TO-DEST customer2]
15.51
[PICK-UP customer1, MOVE-TO-DEST customer1]
15.17
[PICK-UP customer3, MOVE-TO-DEST customer3]
14.96
[..., PICK-UP customer1, MOVE-TO-DEST customer1]
14.86
[..., PICK-UP customer3, MOVE-TO-DEST customer3]
14.54
[..., MOVE-TO-STATION station1, CHARGE station1]
13.00
[..., MOVE-TO-STATION station1, CHARGE station1]
12.39
[..., PICK-UP customer3, MOVE-TO-DEST customer3]
12.19
...
13.37
12.73
12.56
...
12.87
12.22

Figure 6.15: Iteration 12

Finally in Figure 6.16, the last open node is expanded and, since its plan has achieved all goals, it is stored as a solution. As there are no more open nodes, the search process is finished and the best solution is returned.

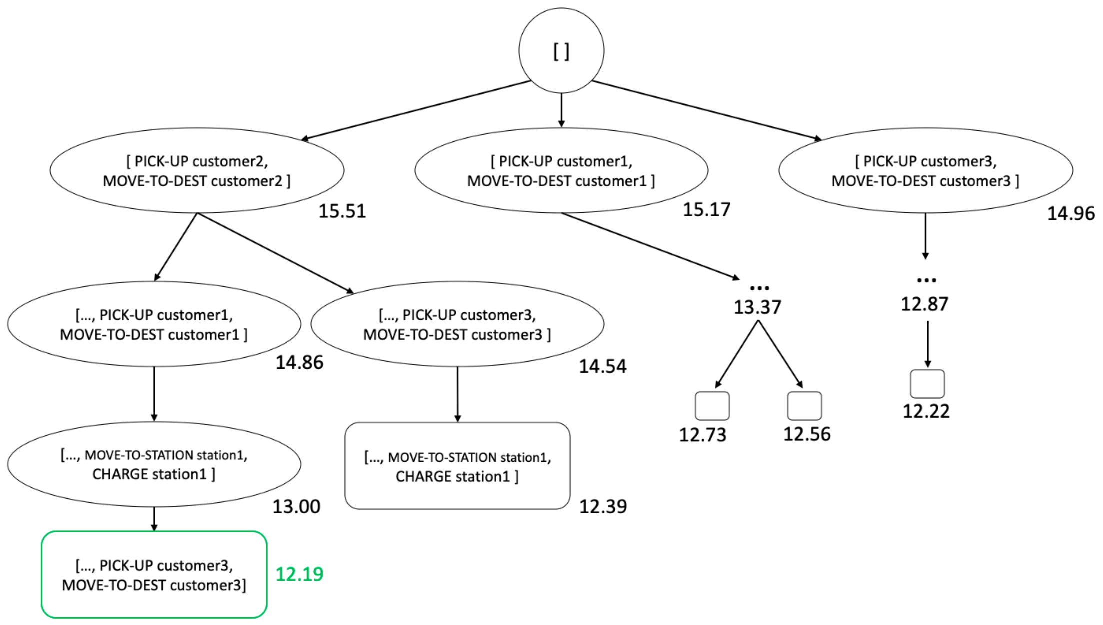

[ ]
[PICK-UP customer2, MOVE-TO-DEST customer2]
15.51
[PICK-UP customer1, MOVE-TO-DEST customer1]
15.17
[PICK-UP customer3, MOVE-TO-DEST customer3]
14.96
[..., PICK-UP customer1, MOVE-TO-DEST customer1]
14.86
[..., PICK-UP customer3, MOVE-TO-DEST customer3]
14.54
...
13.37
...
12.87
12.73
12.56
12.22
[..., MOVE-TO-STATION station1, CHARGE station1]
13.00
[..., MOVE-TO-STATION station1, CHARGE station1]
12.39
[..., PICK-UP customer3, MOVE-TO-DEST customer3]
12.19

Figure 6.16: Iteration 13

A total of 13 planning iterations were needed to find the optimal plan (Table 6.4), generating and expanding 13 nodes. The number of nodes has been reduced thanks to the storage of partial plans, which allowed the planner to use the utility of a solution as lower bound to discard certain nodes from the very beginning of the search process. Without this mechanism, the planner would not have been able to apply any pruning until the first complete solution was found and stored as such.

init time	**actions**	**end time**
0,00	(Agent_A, PICK-UP, customer2)	4,62
4,62	(Agent_A, MOVE-TO-DEST, customer2)	9,97
9,97	(Agent_A, PICK-UP, customer1)	15,40
15,40	(Agent_A, MOVE-TO-DEST, customer1)	21,67
21,67	(Agent_A, MOVE-TO-STATION, station1)	25,26
25,26	(Agent_A, CHARGE, station1)	30,66
30,66	(Agent_A, PICK-UP, customer3)	37,35
37,35	(Agent_A, MOVE-TO-DEST, customer3)	45,26

Table 6.4: Plan obtained by Agent A at the end of the search process. It reports an utility of 12.19.

6.8 Chapter remarks

In this chapter we have described the SimFleet Planner, an ad-hoc planner designed to obtain optimal plans for problems set in our urban traffic domain exclusively.

The design of the planner takes into account the domain's features to perform a computationally cheap search process even in large problems. Also, it includes many techniques that are used to guide the search and prune the search tree, eliminating branches that do not have the potential to improve previously found solutions.

The most relevant attributes of our planner are:

- Use of open goals, that allows transport agents to choose the customers they want to serve according to their interests.
- Table of Goals to avoid conflicts among the agents of the scenario.
- Implementation of a search tree with partial plans in nodes in contrast to plans built action by action. This reduces the branching factor and the expansion in the planning process.
- Storage of partial solutions whenever a goal is solved. The plans that solve more than one goal are incrementally built when it is possible.
- Heuristic search that guides the expansion of nodes of the search tree according to their prospective utility.
- Best solution storage and the use of its utility for pruning the search tree from nodes with no potential to improve the solution.
- Use of a previous plan at the initialization of the planning process, which allows to start from a utility bound to be improved. This makes the planner much more efficient.
- Goal limitation to restrict the amount of open goals that are considered in the planing process. This aids in the obtention of plans for complex scenarios.
- Initial feasible joint plan definition that provides the planner with a previous plan but in the context of the BRPS process.

The planner is intended to be used by the agents of our Best-Response Planning Strategy (BRPS) module (Chapter 7) to find and propose a strategy (plan) that is in best response to the strategies of all other agents. For that, it takes into account what other agents intend to do and plans accordingly to avoid conflicts and, at the same time, maximize utility.

In Chapter 8 we present an empirical evaluation of our planner which is centered around its performance.

CHAPTER 7
Best-Response Planning Strategy

Having the planner presented in Chapter 6, we are able to obtain a planned execution of the transport agents that is guided by their own private interests. However, this alone does not suffice to integrate such rational, self-interested agents in SimFleet. For that, we must ensure that the plan of every transport agent in the simulation scenario is not in conflict with other agents' plans so that the simulation can be executed without flaws. Therefore, we aim to coordinate the actions of the different agents to carry out a complete simulation, while at the same time preserving the self interest of every agent, not enforcing any action on them. For achieving that, we define the simulation as a non-cooperative Multi-Agent Planning (MAP) task, in which the agents will not cooperate but instead coordinate their plans to complete the simulation, serving all customers. The MAP task is solved by a Best-Response Planning Strategy, following the work in [25].

A Best-Response Planning Strategy (BRPS) is a game-theoretic method for non-cooperative MAP tasks. With it, we are able to find a joint plan (a union of plans from different agents) for a group of self-interested agents in domains with congestions and conflicting situations. BRPS makes each agent adapt their plans to every other agent's plans through an iterative process which improves the agent's plan utility. The interactions that cause congestions or conflicts among agents entail an increment of the plan cost (a reduction of utility). Because of that, agents are encouraged to avoid these situations.

BRPS is a general-purpose non-cooperative MAP method and thus can be applied to many domains. In our work, we are applying it to a Urban Traffic domain in which agents are Electric Autonomous Vehicles (EAV), self-interested taxi agents in a smart city, aiming to serve the travel request of the different customers of the simulation scenario, which reports a benefit.

In this chapter we first address the BRPS process from a general point of view, explaining its functioning. Then, we illustrate how we apply it to our urban mobility domain, presented in Section 6.1, with an execution example.

7.1 BRPS process

The BRPS is a process in which an agent a iteratively looks for a plan π^a which is in best response to every other plan in the joint plan Π. At the beginning of the process, an empty joint plan $\Pi = \emptyset$ is created and an arbitrary order is defined among all participant agents. During the process execution, agents must best respond in each iteration. For that, a planning process is used, which can return either a new plan, the same plan as previous iteration or nothing. If the same plan is returned, the agent will preserve it, since it means that it is still in best response towards every other plan. When no agent modifies its plan in a complete iteration, the BRPS has converged at a point in which the joint plan is Pure Nash Equilibrium (PNE).

From an agent's perspective,the BRPS works as follows:

- An arbitrary order between agents is established. Following such order, an initial joint plan is built incrementally: $\Pi = \langle \emptyset, \ldots, \emptyset \rangle, \Pi = \langle \pi^1, \emptyset, \ldots, \emptyset \rangle$, $\Pi = \langle \pi^1, \pi^2, \ldots, \emptyset \rangle, \ldots, \Pi = \langle \pi^1, \pi^2, \ldots, \pi^n \rangle$.
- In one iteration i, agent a executes these steps:
  1. Analyze the utility of its current plan π^a_{i-1} in the joint plan, defining a lower bound for the following search.
  2. A planning process begins to search for a new plan, π^a_i which is in best response to every plan in the joint plan. The planning process is explained in detail in Chapter 6.
  3. If a new plan is returned, the joint plan is updated:

     $$\Pi = \langle \ldots, \pi^a_{i-1}, \ldots, \rangle \rightarrow \Pi' = \langle \ldots, \pi^a_i, \ldots, \rangle$$

     In case no plan with higher utility than the lower bound can be found, the agent keeps its previous plan π^a_{i-1}, since it is in best response.
- When no participant agent changes its plan in a complete iteration, the process has converged and the current joint plan is a PNE.

7.2 BRPS in the Urban Mobility domain

In this section we show how the BRPS is applied to a simulation configuration based on our urban mobility domain. For that, we present a small scenario (Figure 7.1) with 3 transport agents, 3 customers and a charging station, all located in the city of Valencia, Spain.

As the image shows, for each customer, there is a desired destination: *dest. 1* for *customer 1*, *dest. 2* for *customer 2*, etc. The initial autonomy of transport *agents B and C* is at a 100%; however, *agent A* has an initial autonomy of a 33% of the total autonomy. If any agent requires to recharge their batteries at any point, they can do so at *station 1*, centrally located in the urban area.

For the completion of the simulation, transport agents will have to attend all three customers driving them to their destinations. Just by looking at the image, one can intuit that an optimal solution to the simulation, from the point of view of the customers, would be to assign one customer to each agent; particularly customer 1 to agent C, customer 2 to agent A and customer 3 to agent B, since those

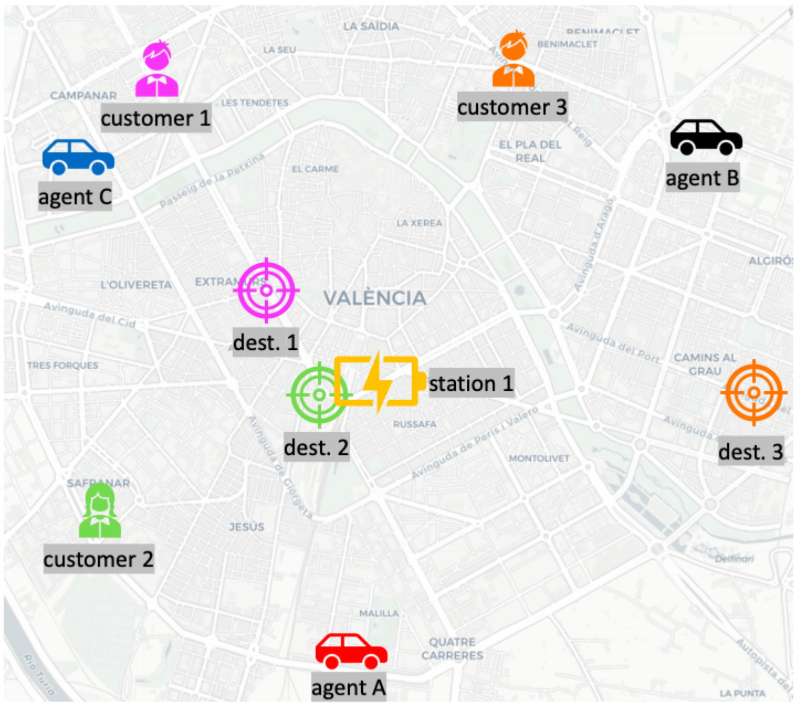


Figure 7.1: Small planning scenario with multiple agents based on the city of Valencia, Spain

pairs are attaching each customer to the transport agent closest to them. However, our system is made to preserve the self-interest of transport agents, so there is no global scheduler that assigns customers to agents in the aforementioned way. Instead, the optimal solution will be accomplished as the BRPS process converges and agents decide that the best plan they can do with respect to each other is to only pick up their closest customer.

Following we present a trace with the execution of the BRPS, depicted in Section 7.1, on the configuration explained above, describing in detail every step.

Initialization

Firstly, a random order is defined among agents. For the sake of simplicity, the order is Agent A → Agent B → Agent C. Then, following that order, agents propose their initial plan, which will be added to the joint plan.

In Table 7.1 all three initial plans are presented, following the structure explained in Section 6.3. As it can be seen, Agent A, who planned in the first place, with no previous information about other agent's plans, has planned to pick up every customer, starting with the one closest to it, and charging after serving two of the customers to have enough autonomy to serve the one left. This plan reports Agent A a utility of 12.19. Once Agent A's plan is added to the joint plan, the Table of Goals gets updated accordingly:

Table of Goals		
goal	agent	pick-up time
customer 1 :	Agent A	15.40
customer 2 :	Agent A	4.62
customer 3 :	Agent A	37.35

Agent B created its initial plan having information about Agent A's plan, which it checks by accessing the Table of Goals. This of course influences its plan. For instance, it plans to serve only customers 1 and 3, since Agent A is picking up customer 2 at time 4.62 and Agent B realized that he can not get to customer 2 before that. However, since Agent A is attending customer 2 first and leaving customers 1 and 3 for later, Agent B can capitalize of that, serving both of them before Agent A has. This plan reports Agent B a utility of 9.57. Then, once Agent B's plan is added to the joint plan, the Table of Goals updates as follows:

Table of Goals		
goal	agent	pick-up time
customer 1 :	Agent B	9.81
customer 2 :	Agent A	4.62
customer 3 :	Agent B	25.06

Finally, Agent C presents its plan as a best response to both Agent A's and B's plans. As it can be seen, it can reach customer 1 before Agent B, which, in turn, makes him get to customer 3 before them as well. Therefore, at the end of the initialization, the Table of Goals presents this information:

Table of Goals		
goal	agent	pick-up time
customer 1 :	Agent C	2.26
customer 2 :	Agent A	4.62
customer 3 :	Agent C	17.51

After the first BRPS iteration, the joint plan is as presented in Table 7.2. It can easily be seen that there are many **customer conflicts**, which make the plan unfeasible. During the BRPS process these conflicts will naturally disappear, achieving a plan that solves the simulation upon convergence.

Agent A		
init time	actions	end time
0,00	(Agent_A, PICK-UP, customer2)	4,62
4,62	(Agent_A, MOVE-TO-DEST, customer2)	9,97
9,97	(Agent_A, PICK-UP, customer1)	15,40
15,40	(Agent_A, MOVE-TO-DEST, customer1)	21,67
21,67	(Agent_A, MOVE-TO-STATION, station1)	25,26
25,26	(Agent_A, CHARGE, station1)	30,66
30,66	(Agent_A, PICK-UP, customer3)	37,35
37,35	(Agent_A, MOVE-TO-DEST, customer3)	45,26
		Utility: 12,19

Agent B		
init time	actions	end time
0,00	(Agent_B, PICK-UP, customer1)	9,81
9,81	(Agent_B, MOVE-TO-DEST, customer1)	16,07
16,07	(Agent_B, PICK-UP, customer3)	25,06
25,06	(Agent_B, MOVE-TO-DEST, customer3)	32,96
		Utility: 9,57

Agent C		
init time	actions	end time
0,00	(Agent_C, PICK-UP, customer1)	2,26
2,26	(Agent_C, MOVE-TO-DEST, customer1)	8,52
8,52	(Agent_C, PICK-UP, customer3)	17,51
17,51	(Agent_C, MOVE-TO-DEST, customer3)	25,41
		Utility: 9,99

Table 7.1: Initial plans

Joint Plan		
init time	actions	end time
0,00	(Agent_A, PICK-UP, customer2)	4,62
0,00	**(Agent_B, PICK-UP, customer1)**	9,81
0,00	**(Agent_C, PICK-UP, customer1)**	2,26
2,26	(Agent_C, MOVE-TO-DEST, customer1)	8,52
4,62	(Agent_A, MOVE-TO-DEST, customer2)	9,97
8,52	**(Agent_C, PICK-UP, customer3)**	17,51
9,81	(Agent_B, MOVE-TO-DEST, customer1)	16,07
9,97	**(Agent_A, PICK-UP, customer1)**	15,40
15,40	(Agent_A, MOVE-TO-DEST, customer1)	21,67
16,07	**(Agent_B, PICK-UP, customer3)**	25,06
17,51	(Agent_C, MOVE-TO-DEST, customer3)	25,41
21,67	(Agent_A, MOVE-TO-STATION, station1)	25,26
25,06	(Agent_B, MOVE-TO-DEST, customer3)	32,96
25,26	(Agent_A, CHARGE, station1)	30,66
30,66	**(Agent_A, PICK-UP, customer3)**	37,35
37,35	(Agent_A, MOVE-TO-DEST, customer3)	45,26

Table 7.2: Joint plan after the initialization. Customer conflicts shown in bold

Process development - Iteration 1

After the initialization, the general BRPS iterations begin. Following the established order, each agent will reevaluate their current plan in the actual joint plan and then search for a new plan which is in best response to every other plan.

Agent A reevaluates his plan. For this, he checks the Table of Goals. The customers which are not being served by him will not count as completed goals, consequently reporting no benefits. Because of that, the utility of his plan has been reduced from 12.19 to 0.78. The latter value, 0.78, is then used as a lower bound for the following planning process. Agent A will now search for a different plan which is in best response to the other agent's actions. The planning is successful and returns a new plan with utility 5.1, presented in Table 7.3. His plan has drastically changed, since the agent has realized that he can only serve one customer before the other agents and even though customer 2 is the closest one to him, apparently serving customer 3 reported a better utility and therefore it chooses to do so. Also, as he does not plan to serve more than one customer, his autonomy is no longer a problem and so he avoids an unnecessary charge, reducing costs.

Agent B has had his plan utility reduced from 9.57 to -1.83. This happened because all of his served customers where effectively "stolen" by other agents, rendering his plan only with costs and no benefits. In case the new utility is below 0, it is not used as lower bound, since we consider that the plan is no longer feasible. Agent C then finds a new plan, with a utility of 9.14 (see Table 7.3).

Finally Agent C, who had its plan utility reduced from 9.99 to 3.80, finds a new plan with utility 8.79 (see Table 7.3). After the change of plans of this iteration, the Table of Goals presents the following information:

Table of Goals

goal	agent	pick-up time
customer 1 :	Agent C	2.26
customer 2 :	Agent C	14.06
customer 3 :	Agent B	3.95

The joint plan obtained after the first iteration is displayed in table 7.4. It still presents some customer conflicts.

Agent A		
init time	actions	end time
0,00	(Agent_A, PICK-UP, customer3)	11,87
11,87	(Agent_A, MOVE-TO-DEST, customer3)	19,77
		Utility: 5,09

Agent B		
init time	actions	end time
0,00	(Agent_B, PICK-UP, customer3)	3,95
3,95	(Agent_B, MOVE-TO-DEST, customer3)	11,85
11,85	(Agent_B, PICK-UP, customer2)	25,37
25,37	(Agent_B, MOVE-TO-DEST, customer2)	30,72
		Utility: 9,14

Agent C		
init time	actions	end time
0,00	(Agent_C, PICK-UP, customer1)	2,26
2,26	(Agent_C, MOVE-TO-DEST, customer1)	8,52
8,52	(Agent_C, PICK-UP, customer2)	14,06
14,06	(Agent_C, MOVE-TO-DEST, customer2)	19,41
		Utility: 8,79

Table 7.3: Agent's individual plans obtained in BRPS iteration 1

Joint Plan		
init time	actions	end time
0,00	**(Agent_A, PICK-UP, customer3)**	11,87
0,00	**(Agent_B, PICK-UP, customer3)**	3,95
0,00	(Agent_C, PICK-UP, customer1)	2,26
2,26	(Agent_C, MOVE-TO-DEST, customer1)	8,52
3,95	(Agent_B, MOVE-TO-DEST, customer3)	11,85
8,52	**(Agent_C, PICK-UP, customer2)**	14,06
11,85	**(Agent_B, PICK-UP, customer2)**	25,37
11,87	(Agent_A, MOVE-TO-DEST, customer3)	19,77
14,06	(Agent_C, MOVE-TO-DEST, customer2)	19,41
25,37	(Agent_B, MOVE-TO-DEST, customer2)	30,72

Table 7.4: Joint plan after iteration 1. Customer conflicts shown in bold

Process development - Iteration 2

Agent A, having had its plan utility reduced from 5,09 to -1,09, finds a new plan in which only serves customer 2, reporting a utility of 4,10. Since every other agent is serving in first place its closest customer, the only thing Agent A can do to also serve one is to do the same.

Agent B has had its plan utility reduced from 9,14 to 4,49 because of the change of plan of Agent A, which makes Agent B unable to serve customer 2. Consequently, its new plan only consists in serving customer 3, which reports him a utility of 5,53.

Similarly to Agent B, Agent C can no longer serve customer 2 and so its previous plan now returns a utility of 4,13, in contrast to the 8,79 that it returned before. Its new plan serves only customer 1 and has a utility of 4,74. The Table of Goals at the end of iteration 2 is:

Table of Goals		
goal	agent	pick-up time
customer 1	Agent C	2,26
customer 2	Agent A	4,62
customer 3	Agent B	3,95

Individual plans after the second iteration can be seen in Table 7.5, whereas the joint plan is presented in Table 7.6. At the end of this iteration, the joint plan presents no conflict, being feasible for the first time. In general, this indicates that convergence has been reached or will be reached soon.

Agent A		
init time	actions	end time
0,00	(Agent_A, PICK-UP, customer2)	4,62
4,62	(Agent_A, MOVE-TO-DEST, customer2)	9,97
		Utility: 4,10

Agent B		
init time	actions	end time
0,00	(Agent_B, PICK-UP, customer3)	3,95
3,95	(Agent_B, MOVE-TO-DEST, customer3)	11,85
		Utility: 5,53

Agent C		
init time	actions	end time
0,00	(Agent_C, PICK-UP, customer1)	2,26
2,26	(Agent_C, MOVE-TO-DEST, customer1)	8,52
		Utility: 4,74

Table 7.5: Agent's individual plans obtained in BRPS iteration 2

Joint Plan		
init time	actions	end time
0,00	(Agent_A, PICK-UP, customer2)	4,62
0,00	(Agent_B, PICK-UP, customer3)	3,95
0,00	(Agent_C, PICK-UP, customer1)	2,26
2,26	(Agent_C, MOVE-TO-DEST, customer1)	8,52
3,95	(Agent_B, MOVE-TO-DEST, customer3)	11,85
4,62	(Agent_A, MOVE-TO-DEST, customer2)	9,97

Table 7.6: Joint plan after iteration 2. It presents no conflicts.

Convergence

During the development of the third iteration, all three agents experiment the same phenomenon. The utility of their plans has not changed, since no plan has any conflict with another. Because of that, the lower bound for the search of a new plan is equal to the original utility of the plan obtained in the previous iteration. The planning process, however, is unable to find a plan that improves the utility, so the agents keep their previous plans, since they are in best response already. As there were no plan changes in a complete iteration, the BRPS process has converged and so it finishes. The joint plan obtained upon convergence (which is the one obtained in iteration 2, presented in Table 7.6) is a feasible, executable plan which accomplishes all global goals (serving all customer travel requests), solving the simulation.

agent	utility
Agent A	4,10
Agent B	5,53
Agent C	4,74

Table 7.7: Utility obtained by each agent after convergence

It can be observed how the BRPS process does take into account the private interests of every transport agent, since they plan to serve as many customers as possible maximizing their own benefits. Nevertheless, at the end of the process, the obtained joint plan is a fair solution from the perspective of the customers, being all of them picked up by their closest transport, consequently arriving to their destinations before. From the point of view of game theory, the general-sum game finished in a win-win situation where every agent serves one customer, obtaining a utility > 0, presented in Table 7.7. The solution is a Pure Nash Equilibrium (PNE), since every agent is doing their best strategy with respect to every other agent's strategy. In this particular example, this is the only PNE; the solution that has been presented will always be obtained by the BRPS process no matter the order in which agents play.

7.3 Integration with SimFleet

The final goal of the proposed BRPS module is to provide SimFleet with the tools to include rational, self-interested agents on their simulations. Both the planner (Chapter 6) and the BRPS modules are implemented outside of SimFleet. Therefore, we need to provide some kind of communication between the output of the BRPS and SimFleet, so that the results of the former can be used for the simulations in the latter.

On the one hand, the BRPS returns an executable joint plan which defines the actions every transport agent in the simulation will perform and in which order. On the other hand, the behavior of SimFleet's agents in the simulations is defined by their *StrategyBehaviour*, as explained in Section 2.1.3, which also determines their methods of communication with other agents. Our aim is to have the agents behaving exactly as their own plan indicates. The agents do not need to communicate anymore, since all coordination has been performed by the BRPS process. Consequently, we designed and implemented the *Extract Plan StrategyBehaviour* for transport agents.

The Extract Plan StrategyBehaviour simplifies the execution of transport agents making them execute the actions in their plan, one after another. Before that, an initialization process loads the joint plan outputted by the BRPS, splits it into the individual plans that compose it, and assigns their plan to every transport agent that will store it as an attribute. Then, the behavior works as follows:

1. The agent extracts the first two instructions of its plan and identifies the type of service they define (customer or charge service).

2. Once the service is identified, the agent initializes the mechanisms already defined in SimFleet for attending a customer travel request or charge in a determined station. Such mechanisms only require the ID of the customer or station to which the actions make reference to, an information that is explicit in the action attributes.

3. After the completion of the service, the agent checks if its plan is empty. If it is, its execution finishes. If it is not, its behavior returns to step (1).

With this, we successfully communicate the BRPS module with SimFleet simulations making them develop in a more realistic way that our algorithm determines; a way that respects the private interests of transport agents.

7.4 BRPS convergence with open goals

When analyzing the convergence of our BRPS process, we ran into an issue. Our approach defines simulations as a non-cooperative Multi-Agent Planning task. To give complete freedom to the transport agents to follow their private interests, we do not assigned them goals but allow them to plan with open goals. Because of that, the convergence of our best-response like process is not guaranteed.

Other authors use models in which their agents have a previously defined fixed set of goals to achieve. However, in our domain we do not assign any goal to any agent; instead, when planning, agents aim to complete as many goals as possible. In other words, our agents plan to pick up and serve as many customers as they can. During the development of the BRPS process, this causes agents to enter in loops, changing the goals they planned to complete in between iterations, thus never achieving convergence. This turned out to be a challenge, since we could not find other works where best-response dynamics were used together with agents with open goals. Therefore, to obtain some results, we developed two methods that aid the BRPS process in its convergence, losing, however, the optimality of the obtained solution.

7.4.1. Blackboard, an amplified Table of Goals

We defined a *blackboard*, a globally accessible table. It contains one row per customer in the simulation scenario. In turn, each customer has one entry per transport agent where we store a list with tuples (agent, pick-up time), in a similar way than in the Table of Goals (explained in Section 6.4.1).

At the end of every BRPS iteration, the entries of the Table of Goals are copied into the corresponding entry of the Blackboard. For instance, if the Table of Goals indicates that "customer 1" is picked up by agent "taxi 1" at time "3.5", the tuple ("taxi 1", "3.5") will be appended to the "customer 1" row, "taxi 1" entry of the Blackboard. Consequently, each BRPS iteration the information in the Blackboard increases.

The Blackboard is then used during the planning process as another mechanism to check if a customer is reachable, thus restricting the planning alternatives even more. In contrast with the Table of Goals, that shows which agent serves which customer *in the current joint plan*, the Blackboard shows the same information but about *every previous joint plan of the process*. Because of that, the current planning agent can check if a customer that is reachable at the moment could be potentially stolen by another agent in future iterations. In other words, if an agent has previously planned to pick up that very same customer and it was reaching it before the current agent can, it could plan to pick it up again, effectively stealing it from the current agent. Of course, even though another agent could steal such customer it does not imply that it will. However, for the sake of achieving a convergence, if an agent, during its planning process, checks that a customer it is planning to serve may be stolen by another agent, it will discard such partial plan.

With the use of the Blackboard, the planning process gets more restricted and the agents are less encouraged to change their plans and, consequently, their goals in between BRPS iterations. In this way, we "force" an artificial convergence which, even though it is not optimal, gets an equilibrium solution and, most importantly, a feasible joint plan that can be executed in SimFleet as a complete simulation.

7.4.2. Goal fixing BRPS algorithm

The BRPS process with the use of the Blackboard (see Section 7.4.1) was useful to study how much time and iterations the procedure took to converge with different simulation scenarios. However, in terms of solution quality, the joint plan that it obtained could be improved.

Aiming to obtain better BRPS solutions, we designed and implemented an algorithm that allows the agents to plan in accordance to their interests but fixes one goal each BRPS iteration, thus forcing the convergence.

The *goal fixing* BRPS algorithm is a process in which the agents get certain goals assigned to them iteratively. Every iteration, each agent can be assigned a goal from among those completed in their proposed plan. Then, in following iterations, the agent's proposed plan will always start by completing their assigned goals and then be completed freely by the agent's planning process.

The goal assigned to each agent is the first not previously assigned goal that appears in their proposed plan. We are then assuming that such goal is the one that arouses the most interest and, therefore, brings the greatest benefit to the agent. Of course, this is not always true, so, again, this method loses the optimality of the solution.

The previously described procedure may present a conflict, however, if two or more agents get the same goal fixed in an iteration. The algorithm takes that into account and only one of them keeps the goal; specifically, the agent that achieves the goal before in terms of time (the agent that picks the customer up earlier). When an agent that was in conflict loses its fixed goal, it plans again but respecting the goals fixed for every other agent, thus obtaining a new plan in which the goal to fix does not present any conflicts.

Therefore, each iteration the agents propose a plan and get a goal fixed, if possible. This process ends when all goals have been assigned; i.e: when all customers have been distributed between the transport agents. Following, we describe the Goal fixing algorithm in detail.

Initialization

- Every agent proposes an initial plan from scratch. In contrast to our BRPS process, the agents do not plan in turns but in parallel, thus not considering each others' plans. These plans are obtained with complete freedom regarding agent self-interest, as there are no fixed goals yet.
- Once the plans have been obtained, the procedure to fix a goal (customer) for each agent begins. The first goal achieved by each agent is compared:
  - If there are no conflicts (no repeated goals), the first goal completed in each plan is assigned and fixed to its corresponding agent.
  - If two or more agents have a repeated first goal, the goal will be assigned to the agent that achieves it earlier.
  - The agents that were in conflict and are left without a fixed goal enter the conflict solving phase:
    - The Table of Goals is updated with the fixed goals stored in the respective position.
    - Then, the agents with no fixed goal plan again from scratch, but with the previous information given by the Table of Goals. Therefore, the obtained plan respects the goals assigned to other agents and presents no conflicts with them.
    - The first goals obtained by the newly proposed plans are fixed to their respective agents and the Table of Goals is updated.
    - Finally, if after this procedure one or more agents were still in conflict and had no fixed goal, the process would be repeated until they did.
- After the initialization, each agent has one goal assigned and fixed. Then the general iterations of the process begin.

General iteration

- In a general iteration of the process the agents propose a plan which is built in two steps:

  1. Planning process to complete the assigned goals. It does not respect the agents self-interest since we force them to plan for fixed goals.
  2. Planning process which begins from the partial plan obtained in the previous step. In this case the agent plans freely to complete as many open goals as it can (like in the BRPS process) respecting, however, the goals assigned to other agents, which are presented in the Table of Goals. This process does take into account the private interests of the agent.

- The agents have computed a plan which has been partially obtained considering their own self-interest. Following, the goal fixing procedure begins and any conflicts that arise are solved.

- In a general iteration the goal to fix is not the first goal completed in the proposed plan but the first goal achieved by the plan which is not a previously fixed goal. The Table of Goals is updated with the fixed goals as explained in the initialization.

- In the conflict resolution phase one or more agents may not be able to fix a goal if all the process goals have already been assigned. This will happen if the number of customers is not evenly divided between the number of transport agents.

- After a general iteration, the stop condition of the algorithm is checked. If every goal has been assigned, the process goes on to the finalization. If not, there will be another general iteration.

Finalization

- When this point is reached every agent has their goals assigned. In addition, the distribution of the goals is fairly uniform among agents. Then, to complete the algorithm and return a feasible joint plan, the agents perform another planning process in which they only plan to complete their assigned goals (since every goal is assigned, they would not be able to plan further either way).

The joint plan obtained from the Goal fixing BRPS algorithm presents no conflicts and can therefore be turned into a complete SimFleet simulation. Nevertheless, this solution is neither an equilibrium nor optimal. It is, however, a near-optimal solution which improves the equilibria obtained by the BRPS process with the Blackboard.

7.5 Chapter remarks

In this chapter we have described the BRPS module, used to introduce rational, self-interested agents in SimFleet simulations so as to achieve more realistic executions. For that, the SimFleet's simulation is redefined as a Multi-Agent Planning (MAP) task, in which the global goal is to solve the simulation, which is achieved by serving the travel requests of every customer. The module is then in charge of coordinating the actions of the transport agents to complete the global goal ensuring, at the same time, that their interests are preserved. In this case, as transport agents are modeled as electric autonomous taxis, their interest is to maximize their profit, which implies serving as many customers as possible.

The coordination of agent's execution is done by a game-theoretic process that defines a congestion game in which the agent's are players. For such game, the strategies are individual plans, obtained by our ad-hoc planner (Chapter 6). An individual plan describes the actions that a single agent intends to perform to maximize its utility while solving part of the MAP task. By uniting every individual plan we obtain a joint plan, which is a solution to the MAP task. However, the joint plan must be executable; it must present no conflicts among the agents of the simulation. To ensure the obtention of a feasible joint plan, the game is developed by a best-response process.

The best-response dynamics, an iterative process in which each agent proposes its best plan considering the other agents plans, ensures that conflicts are avoided by penalizing the agent's utility when they are present. During the development of the game, agents propose their strategies, in turns, always as a best response to the strategies of all other participant agents. With the aid of the planner, the strategies of the agents are not only optimal with respect to other agents' strategies but also avoid conflicts.

Once the best-response process converges, an executable joint plan has been obtained. This plan is then performed by transport agents in the SimFleet simulation using their Extract Plan StrategyBehaviour, effectively obtaining a simulation in which the self-interest of the agents is taken into account.

After the implementation of the BRPS module we discovered that the use of open goals made it harder for the process to converge. Aiming to palliate such behavior and to obtain feasible solutions, we designed and implemented the Blackboard and the Goal fixing BRPS algorithm, both methods that aid in the convergence of the BRPS process and obtain near-optimal, executable joint plans.

In the following Chapter 8 we analyze the performance of both the planner and the whole BRPS process by running some experiments with problems that vary in complexity.

CHAPTER 8
Experimentation

In this chapter we present some experiments performed with the ad-hoc planner (Chapter 6) and the Best-Response Planning Strategy (BRPS) process (Chapter 7) aiming to evaluate their efficiency and study how their performance varies according to the size and complexity of the problem they have to solve.

We first evaluate the planner individually, measuring the effect of the different pruning and speedup techniques we have developed. Then, we evaluate the BRPS process, comparing the solution it obtains to the solutions obtained by SimFleet.

Regarding the contributions described in Chapters 4 and 5, we present experimentation on them in Sections 4.3 and 5.3, respectively. We decided to focus this chapter on the Planner-BRPS module as it the main contribution of our thesis.

8.1 Planner performance

To show how the planner behaves with different problem sizes, we defined five different simulation scenarios which vary in complexity, presented in Table 8.1. We defined three metrics, which allow us to compare various planner executions. The metrics are the number of nodes generated in the search process (# Nodes), the maximum size of the open node priority queue (Max. queue) and the time (Time (s)) the whole planning process took, in seconds. Comparing the first two values, we can get an idea of the memory consumption of each planning process. Regarding the time, although it may vary depending on the machine that executes the algorithm, we can still see how the problem size increases the duration of the search. All the presented results were obtained with a machine running Windows 10 OS, with an Intel Core i7-9750H 2.60GHz processor and 16GB of RAM.

Problem nº	Taxi Agents	Customers	Stations
1	3	3	1
2	3	6	2
3	5	10	3
4	10	30	5
5	20	60	8

Table 8.1: Problem instances defined to test the effect of different scenario sizes over the computational power of the planner.

8.1.1. Effect of the pruning methods

In this first test, we studied the influence that each of the pruning and planning search speedup methods have over the time and memory consumption of the planner. For that, we obtained different plans for the problem instance nº2 (see Table 8.1), which defines a simulation scenario with 3 electric autonomous taxi agents, 6 customers and 2 charging stations, alternating the methods that were active in each execution.

The methods considered for evaluation were the use of an heuristic evaluation function (Section 6.4.3) (Heuristic) to guide the search process; the storage of partial solutions (Section 6.5.2) and the application of best solution pruning (Section 6.5.1) (Pruning); the existence of a previous plan (Section 6.5.3) (Prev. Plan) which could also be an initial feasible joint plan (Section 6.6.2) (Feasible J.P.); and finally the limitation of the amount of customers an agent planned to pick up (Section 6.6.1) (Goal Lim.).

The results of the different executions are presented in Table 8.2. Note that for the rows where Prev. Plan is "No", the plan is obtained by an agent with no previous information (empty Table of Goals, no previous plan) thus taking considerably more time to search for the optimal plan. On the other hand, where Previous Plan is "Yes", the planner is executed after the initial iteration of the BRPS process, where the agent does have a previous plan and the Table of Goals contains information. With a Goal Lim. of a 60%, the agent plans to pick up 3 customers. Finally, when an initial feasible joint plan is calculated the agent does have previous information even in the first planning process.

Heuristic	Pruning	Prev. Plan	Goal Lim.	Feasible J.P.	# Nodes	Max. queue	Time (s)
No	No	No	No	No	192135	25	69.96
Yes	No	No	No	No	40602	25104	12.99
Yes	Yes	No	No	No	17016	10155	12.98
Yes	Yes	No	60%	No	8848	4062	2.52
Yes	Yes	Yes	No	No	34	19	0.02
Yes	Yes	Yes	No	Yes	27	13	0.02

Table 8.2: Comparison of the planning process' memory and time consumption according to the pruning and planning speedup methods applied to the plan search.

The effect of using an heuristic function to guide the plan search can be seen comparing the first and second rows of Table 8.2. Just by using it, the amount of generated nodes is reduced almost 5 times. The difference in the maximum size of the open node queue is striking, although it is simply because the heuristics keep more nodes open by considering their potential as a future solution. Anyhow, the time required to find a plan is between 5 and 6 times lower which achieves a drastic speedup of the process. In addition, if we also include pruning with storage of partial solutions the number of nodes is reduced even more, reaching about 11 times less than in the base case (first row). In the fourth row we apply a goal limitation of a 60%, forcing the agent to plan only to pick up 4 customers. This method, which may seem exaggerated for problem instance nº2, is necessary for the calculation of the initial plans in the first round of the BRPS for bigger problems. Comparing the reduction in both generated nodes and planning time obtained with this method with respect to using only heuristic and pruning, it can be seen how more complex scenarios get benefited from it. However, the greatest improvement comes from the inclusion of previous planning information such as the Table of Goals or the previous plan's utility. With that we achieve the fastest speed at a mere 20 milliseconds, generating just a few tens of nodes. This can be seen in the fifth and sixth rows of Table 8.2, where having a previous plan or an initial feasible joint plan has an equivalent effect over the planner efficiency.

Since our planner is designed to be integrated in a BRPS process where agents propose plans, the most relevant result is the presented in the fifth row of Table 8.2, as it is the general situation in which the planner will be executed. Except in the first iteration of the game, the agents generally have a previous plan that limits their search using its utility as a lower bound to improve. In these cases, the planner obtains the optimal plan with very little time and memory consumption.

8.1.2. Size of the scenario

After seeing the effects of the different speedup methods over the same problem, we tested the effect of the size of the planning scenario. Because of the planner's design, the ramification of its search process is mostly affected by the number of customers and stations in the scenario. The number of agents does not affect the planner directly since the planning process is individual for each agent. Instead, by having more agents the planner can actually be indirectly speeded up because the Table of Goals presents more restrictions. However, in general, an increase in transport agents is matched by an increase in customers, which causes the described effect to fade.

To evaluate each problem instance (Table 8.1), we obtained a plan for solving them without any previous information (empty Table of Goals, no previous plan). We limited, however, the amount of goals a customer planned to achieve as the problem size grew. The results are presented in Table 8.3.

Plotting the results (see Figure 8.1), we can see how the computational power consumption of the planner increments exponentially with the complexity of the problems. For the goal limitation, we always try to use values that make each agent pick up more than $\frac{\#customers}{\#agents}$ customers. This is done for problem instances nº1 to nº3. However, the size of problem instances nº4 and nº5 is such that we had to limit the goal number to exactly $\frac{\#customers}{\#agents}$, which in those cases was 3 customers per transport agent. Even with that, the variability of problem

Problem nº	Goal Lim.	Feasible Joint Plan	# Nodes	Max. queue	Time (s)
1	No	No	40	19	0.02
2	60%	No	8848	4062	2.52
3	40%	No	276799	101232	91.13
4	10%	No	876975	405670	449.60
5	5%	Yes	345124	226307	1568.04

Table 8.3: Comparison of the planning process' memory and time consumption according to the problem size.

instance nº5 was so high that we needed to include an initial feasible join plan to obtain another plan in a reasonable time (30 minutes). In the planning process for problem instance nº5, even though less nodes are generated, the search process takes almost 3 times longer than for problem instance nº4.

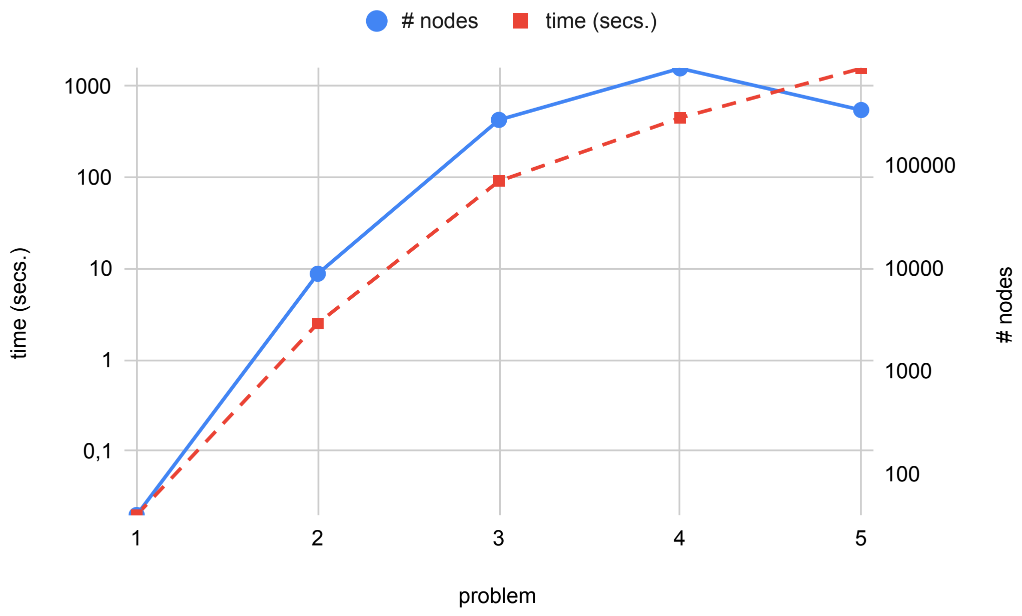


Figure 8.1: Log-scale plot of generated nodes and time according to problem instance. Data extracted from Table 8.3.

8.2 BRPS convergence

For this first experiment we study the effect of the problem complexity over the convergence iteration and time of the BRPS process.

Using the BRPS process with the Blackboard (Section 7.4.1) we managed to perform a comparison of convergence iterations and time according to problem instance size. For that, we executed the simulation scenarios presented in Table 8.1 with different configurations of the procedure. The results can be seen in Table 8.4.

Problem nº	Goal Lim.	Initial Joint Plan	Convergence it.	Time (s)
1	No	No	4	0.04
2	No	No	5	0.69
3	40%	No	6	4.00
4	10%	No	8	1108.22
5	5%	Yes	-	-

Table 8.4: BRPS process convergence time according to problem size

The BRPS process time is closely related to the planning duration, since it is the most long lasting part of the procedure. The convergence iteration (Convergence it. in Table 8.4) has more to do with the complexity of the interactions between agents' individual plans. A higher number of conflicts entails a higher convergence iteration. As for the tested problem instances, the convergence iteration does not vary significantly because of the use of the Blackboard, that stops most joint plan conflicts from arising. We could not analyze the convergence for problem instance nº5 because of its high number of alternatives when planning, which made the process too time consuming for the machines to which we had access.

8.3 SimFleet vs BRPS: solution comparison

In this experiment we executed two simulations, one using the default SimFleet and another which follows a solution obtained with our BRPS module. The metrics that SimFleet offers to evaluate the simulation are the distance travelled by the transport agents and the customer waiting and total time, in seconds. The customer waiting time refers to the time elapsed from the beginning of the simulation until the customer gets picked up. The total time refers to the time elapsed from the beginning of the simulation until the customer reaches its destination. In the solutions obtained by our planner, the waiting time is equivalent to the customer pick-up time, meanwhile the total time is the same as the end time of MOVE-TO-DEST type actions.

With the Goal fixing BRPS algorithm working (Section 7.4.2), we were able to obtain a feasible joint plan for problem instance nº2 (see Table 8.1) with our BRPS module. Then, such plan was followed by SimFleet agents by means of their Extract Plan StrategyBehaviour (Section 7.3), thus allowing us to obtain the simulation metrics. On the other hand, we loaded problem instance nº2 on the default SimFleet simulator and executed a simulation over it.

The simulation metrics were compared from the point of view of both customers and transport agents. For the customers, we assume that a shorter waiting time indicates a better outcome. As for transport agents, we assume that more customer services and less traveled distance indicate a better outcome; i.e: one that reports more utility. These comparisons can be seen in Tables 8.5 and 8.6.

Table 8.5 shows the waiting and total times for each customer in problem instance nº2 both in the solution obtained by SimFleet and the one obtained by our

	SimFleet		BRPS module	
Customer	Waiting Time	Total Time	Waiting Time	Total Time
customer 1	14.57	17.99	5.95	13.07
customer 2	24.76	30.87	3.26	15.59
customer 3	25.92	33.47	21.05	35.35
customer 4	44.64	52.27	15.77	30.72
customer 5	10.90	19.79	21.03	38.55
customer 6	9.23	14.08	6.71	17.19
Total	130.02	168.47	73.77	150.47
Avg.	21.67	28.08	12.30	25.08

Table 8.5: Comparison of simulation solutions obtained by default SimFleet and our BRPS module from the point of view of the customers. All times are indicated in seconds

BRPS module. It can be seen how the average waiting time of the customers is reduced in 9 seconds, thus obtaining a better result from a global perspective. Therefore, the customer satisfaction in a system where the BRPS module is used to schedule the transport agent assignments would be higher.

	Simfleet		BRPS module	
Agent	Served customers	Utility	Served customers	Utility
taxi 1	customer1, customer3	7.62	customer2, customer3	8.04
taxi 2	customer2, customer5	7.68	customer1, customer4	8.29
taxi 3	customer4, customer6	6.51	customer6, customer5	7.86

Table 8.6: Comparison of simulation solutions obtained by SimFleet and our BRPS module from the point of view of transport agents

Table 8.6 shows the customers that are picked up by each transport agent using SimFleet and our BRPS module. As it can be seen, the SimFleet solution achieves a uniform distribution of the customer services as well. However, the customer assignments are performed more efficiently by our BRPS module, reporting more utility to the transport agents.

The simulations that follow joint plans obtained by our BRPS module report, in general, more benefits to each individual transport agent and reduce the average waiting time of the customers.

8.4 Evaluation remarks

In this chapter we have shown various metrics that characterize the performance of our Planner and Best-Response Planning Strategy module.

The ad-hoc planner, designed to obtain plans for problems set in our Urban Traffic domain, has proved to be a computationally efficient algorithm. Although, as any other planner, its performance is greatly influenced by the problem's size, by combining all pruning techniques and search speedup methods we can obtain optimal plans in a relatively short time. When the planner is working integrated with the BRPS process, as it is supposed to be, its time and memory consumption reach the lowest point thanks to the use of previous process information.

As for the BRPS process, we have managed to improve SimFleet simulations, making them more realistic as transport agents act following their own self-interest. Even though some of the more complex problems have trouble to converge, we developed two mechanisms that, used during the development of the process, make it obtain a feasible joint plan. That said, we will without a doubt research further into the convergence of best-response like processes when working with open goals as well as develop new methods that allow us to obtain better equilibria in the final joint plans.

The Planner and BRPS module has enhanced SimFleet's potential, offering a new way of simulation which can be used as a tool to further research into the optimization of the urban traffic.

CHAPTER 9
Conclusions

In this chapter we first asses the work set out in this thesis. Then, we discuss on various extensions that we intend to complete in the future, continuing the work that began with the thesis. We also mention different publications made during the development of the thesis, as well as the research projects that give meaning to them. Finally, we comment the relationship of the research topics developed in our work with the subjects studied in our master's degree.

9.1 Assessment

In this work we have designed and implemented three different modules that work in accordance to SimFleet simulator to enhance its properties. Following, we asses the work done, commenting on the strengths and weaknesses of each contribution.

With the creation of the Load Generators (Chapter 4), we have provided a tool for SimFleet users that allows them to easily define complex simulation configurations, providing a means to create different distributions of agents in the scenario. In addition, the use of the generators can make the simulations it prepares more realistic by giving the agents origin and destination points as well as movement based on real-world data. Nevertheless, we would like to improve the generators by studying better ways of distributing the agents and taking into account more parameters to take more advantage of the real-world data it uses.

The design and inclusion of a system to simulate free-floating carsharing fleets in SimFleet (Chapter 5) has broadened the functionalities of the simulation software, creating a precedent for the future development of more types of fleets.

Finally, we have researched the use of rational, self-interested agents ultimately designing an external module that builds a multi-agent planning task from a SimFleet configuration file and solves the simulation using transport agents that follow their private objectives (Chapters 6 and 7). This involved the creation of an ad-hoc planner that obtains optimal plans for the transport agents that take into account other agent's plan to both avoid conflicts and maximize the utility. For a correct completion of the MAP task we needed to achieve an equilibrium, a stable solution from which no agent had incentive to deviate. For that, we designed a Best-Response Planning Strategy, following the work in [25].

The solutions obtained by this process were later integrated in SimFleet thanks to the implementation of an agent behavior specifically designed to follow the actions indicated by a plan. In this way, we achieved the inclusion of self-interested agents in SimFleet simulations which accomplishes the goal of making the simulation more realistic for the use of taxi or electric autonomous agents.

It is important to mention, however, that the used approach did not work exactly as we expected for the problems in our domain. By defining MAP tasks with open and not fixed goals, the convergence of the BRPS process is not guaranteed under certain conditions which are more common in complex simulation scenarios. This led us to make adjustments (Blackboard, Goal fixing BRPS algorithm) that palliate the issue in exchange of losing the optimality of the solutions. Nevertheless, given the investigative nature of our work, we do not consider this a failure, but rather a learning experience. In the future we plan to analyze this problem in a theoretical way to better understand the conditions that guarantee convergence in best-response dynamics.

All in all, the objectives set for this thesis have been accomplish. We must note, however, that the challenges that brings mobility and in particular urban mobility, are far from being solved and that we intend to keep researching and working towards finding solutions.

9.2 Future work

SimFleet has been greatly enhanced with our work. Nevertheless, there is always room for improvement and we intend to keep working on it, broadening its functionalities to obtain a more competent and user-friendly simulation software.

More specifically, we would like to adapt the Planner and Best-Response Planning Strategy module to other types of urban fleets such as carsharing systems. In addition, we would like to improve such system and add new ones so as to be able to build very complete simulation scenarios with different types of systems in place.

Finally, we would like to point out that the work presented in this thesis is a small part of a bigger project focused on intelligent and sustainable mobility. The compilation of a thesis is, after all, a finite process and at some point we had to consider it finished. However, our intentions are to continue researching game theory, negotiation, coordination and planning techniques for multi-agent systems, aiming to improve mobility or any other areas where our work could be applied.

9.3 Research activities

During the development of this thesis, we have also dedicated ourselves to different research activities, which have resulted in a couple of publications. These publications are part of two research projects that focus on improving urban mobility through the use of artificial intelligence. The activities previously mentioned are presented below.

9.3.1. Publications

- Pasqual Martí, Jaume Jordán, Javier Palanca, and Vicente Julian. **Load generators for automatic simulation of urban fleets**. In *Highlights in Practical Applications of Agents, Multi-Agent Systems, and Trust-worthiness. The PAAMS Collection*, pages 394–405, Cham, 2020. Springer International Publishing.
- Pasqual Martí, Jaume Jordán, Javier Palanca, and Vicente Julian. **Free-floating carsharing in SimFleet**. Accepted for *IDEAL'2020 Main Track. 21st International Conference on Intelligent Data Engineering and Automated Learning*, 2020.

9.3.2. Projects

- MINECO/FEDER RTI2018-095390-B-C31 "Hacia una mobilidad inteligente y sostenible soportada por sistemas multi-agentes y edge computing".
- UPV PAID-06-18 (SP20180184) project "Técnicas inteligentes para optimización de la localización de estaciones de recarga de vehículos eléctricos y mejora de la movilidad en ciudades".

9.4 Connection with the taken studies

During the coursing of the master's degree in which this thesis was developed there were many subjects that focused on related topics:

- In *Intelligent Planning* we revisited the basis of automated planning and learned the techniques applied on planner software to solve such computationally complex tasks in a reasonable time, obtaining near-optimal solutions.
- In *Multi-agent Systems* the basis about intelligent agents and their interactions were introduced. We developed the skills to program environments in which many agents interact, taking into account the necessary coordination.
- In *Tools and Applications of Artificial Intelligence* multi-agent systems applied to real-world commercial processes were presented. Also, multi-agent simulators and some of their applications were commented.
- In *Automated Negotiation* we learned about game-theory basics, a topic we then researched thoroughly for our work. In addition, we studied techniques used by negotiation agents to reach mutually beneficial agreements.

Bibliography

[1] Michael J Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(2):115–152, 1995.

[2] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 21–35. Springer, 1996.

[3] Flávia F Feitosa, Quang Bao Le, and Paul LG Vlek. Multi-agent simulator for urban segregation (masus): A tool to explore alternatives for promoting inclusive cities. *Computers, Environment and Urban Systems*, 35(2):104–115, 2011.

[4] Javier Palanca, Andrés Terrasa, Carlos Carrascosa, and Vicente Julián. Simfleet: A new transport fleet simulator based on mas. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 257–264. Springer, 2019.

[5] Miguel Escrivà, Javier Palanca, and Gustavo Aranda. A jabber-based multi-agent system platform. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, page 1282–1284, New York, NY, USA, 2006. Association for Computing Machinery.

[6] P. Saint-Andre. Extensible messaging and presence protocol (xmpp): Core. RFC 6120, RFC Editor, March 2011.

[7] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory & Practice*. Elsevier, 2004.

[8] Mathijs De Weerdt and Brad Clement. Introduction to planning in multiagent systems. *Multiagent Grid Systems*, 5(4):345–355, 12 2009.

[9] Edmund H. Durfee. *Distributed Problem Solving and Planning*, pages 118–149. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[10] Alejandro Torreño, Eva Onaindia, and Óscar Sapena. Fmap: Distributed cooperative multi-agent planning. *Applied Intelligence*, 41(2):606–626, 9 2014.

[11] John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 2007.

[12] Roger B Myerson. *Game Theory*. Harvard University Press, 2013.

[13] Martin J Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[14] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[15] Adam Brandenburger. Cooperative game theory. *Teaching Materials at New York University*, 2007.

[16] Donald B Gillies. Solutions to general non-zero-sum games. *Contributions to the Theory of Games*, 4(40):47–85, 1959.

[17] John Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.

[18] Raz Nissim and Ronen I. Brafman. Cost-optimal planning by self-interested agents. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, pages 732–738, 2013.

[19] Pieter Buzing, Adriaan Ter Mors, Jeroen Valk, and Cees Witteveen. Coordinating self-interested planning agents. *Autonomous Agents and Multi-Agent Systems*, 12(2):199–218, 2006.

[20] Jan Hrnčíř, Michael Rovatsos, and Michal Jakob. Ridesharing on timetabled transport services: A multiagent planning approach. *Journal of Intelligent Transportation Systems*, 19(1):89–105, 2015.

[21] Jaume Jordán and Eva Onaindía. Game-theoretic Approach for Non-Cooperative Planning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1357–1363, 2015.

[22] Jaume Jordán, Alejandro Torreño, Mathijs De Weerdt, and Eva Onaindia. A non-cooperative game-theoretic approach for conflict resolution in multi-agent planning. *Group Decision and Negotiation*, 2020.

[23] Anders Jonsson and Michael Rovatsos. Scaling up multiagent planning: A best-response approach. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, 2011.

[24] Robert W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973.

[25] Jaume Jordán, Alejandro Torreno, Mathijs De Weerdt, and Eva Onaindia. A better-response strategy for self-interested planning agents. *Applied Intelligence*, 48(4):1020–1040, 2018.

[26] Pasqual Martí, Jaume Jordán, Javier Palanca, and Vicente Julian. Load generators for automatic simulation of urban fleets. In *Highlights in Practical Applications of Agents, Multi-Agent Systems, and Trust-worthiness. The PAAMS Collection*, pages 394–405, Cham, 2020. Springer International Publishing.

[27] Pasqual Martí, Jaume Jordán, Javier Palanca, and Vicente Julian. Free-floating carsharing in simfleet. Accepted for IDEAL'2020 Main Track. 21st International Conference on Intelligent Data Engineering and Automated Learning, 2020.

[28] Jaume Jordán, Javier Palanca, Elena Del Val, Vicente Julian, and Vicente Botti. A multi-agent system for the dynamic emplacement of electric vehicle charging stations. *Applied Sciences*, 8(2):313, 2018.

[29] Javier Palanca, Jaume Jordán, Javier Bajo, and Vicent Botti. An energy-aware algorithm for electric vehicle infrastructures in smart cities. *Future Generation Computer Systems*, 108:454 – 466, 2020.

[30] Elena del Val, Javier Palanca, and Miguel Rebollo. U-tool: A urban-toolkit for enhancing city maps through citizens' activity. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 243–246. Springer, 2016.

[31] Richard Katzev. Car sharing: A new approach to urban transportation problems. *Analyses of Social Issues and Public Policy*, 3(1):65–86, 2003.

[32] Monitor Deloitte. Car sharing in europe–business models, national variations, and upcoming disruptions. *Dosegljivo: https://www2. deloitte. com/content/dam/Deloitte/de/Documents/consumer-industrial-products/CIP-Automotive-Car-Sharing-in-Europe. pdf*, 2017.

[33] Jörg Firnkorn and Martin Müller. What will be the environmental effects of new free-floating car-sharing systems? the case of car2go in ulm. *Ecological economics*, 70(8):1519–1528, 2011.

[34] Ngone Arame Niang, Martin Trépanier, and Jean-Marc Frayret. A multi-agent simulation approach to modelling a free-floating carsharing network. 2020.

[35] Vassio, Luca; Giordano, Danilo; Mellia, Marco; Cocca, Michele . "Data for: "Free Floating Electric Car Sharing Design: Data Driven Optimisation". Anonymized datasaset of 2 months of trips of car sharing users in the city of Turin, Mendeley Data, v1. `http://dx.doi.org/10.17632/drtn5499j2.1`, 2019.