Rafael Filipe Carreira Henriques

TAILORED FIELD-SERVICE OPTIMIZATION

UNIVERSIDADE D COIMBRA
1 2 9 0

UNIVERSIDADE D
COIMBRA

Rafael Filipe Carreira Henriques

TAILORED FIELD-SERVICE OPTIMIZATION

Internship Report in the context of the Master in Informatics Engineering, Specialization in Software Engineering, advised by Professor Pedro Abreu and presented to Faculty of Sciences and Technology / Department of Informatics Engineering.

January 2021

Faculty of Sciences and Technology

Department of Informatics Engineering

Tailored Field-Service Optimization

Rafael Filipe Carreira Henriques

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering advised by Prof. Pedro Abreu and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.

January 2021

UNIVERSIDADE D
COIMBRA

Acknowledgements

I would first like to thank both my advisors for the time they invested in giving me valuable advices and be available to answer any questions. Prof. Pedro Abreu for his unvaluable feedback and help me to focus on the most important aspects. Dr. Bruno Cabral for having guided me closely throughout this work, and beyond the role of advisor, the knowledge transmitted in the area of software engineering.

My profound thanks to *Sentilant* for this internship opportunity that allowed me to work with a great team of dedicated people that taught me a lot. In addition, I loved the opportunity to redirect the internship course at the beginning of the second semester to develop a solution for a real client and become a member of the team.

A huge thanks for my family and my friends, specially my dear girlfriend, that calmed me down mainly in the worst moments when I got stressed out for spending many time trying to find answers to some problems that arose during the course of this work.

Abstract

Despite many field-service companies plan their tasks manually, there is software that helps companies managing and optimizing their resources employed on field service operations. In Field-Service Management, there is a demand for solutions tailored to the client's needs, one of the solutions sought is the creation of optimized work plans for logistics and field-service. A work plan defines tasks to be performed by human resources on the field that can be evaluated with Key Performance Indicators, thus allowing to measure the quality of a work plan when compared with other plans.

In the context of this internship, *Sentilant* develops and sells a Field-Service Management service. One of the main objectives of this service is generating work plans for his clients. The different clients of *Sentilant* have specific needs for their businesses, so there is a need for solutions tailored for those clients and new ones. To build a tailored planning solution, *Sentilant* iteratively develops a model based on the client's feedback, but it is time-consuming to build. The company uses a solver to implement the models, but seeks for alternatives that could reduce the development time-effort, while producing similar or better work plans.

To address this issue, the main goal of this internship is developing a multi-tenant planning system that serves optimized work plans, containing performance indicators to let the clients evaluate their solutions, and intervene in the optimization to influence the resulting plans.

An extensive comparison of 3 solvers to develop models was carried out, as a conceptual analysis was not enough to take conclusions a model was implemented and compared between the solvers (passengers model). Then, using existing software components at *Sentilant*, a planning system API was developed to provide a proposal of optimized work plans for a fuel transports problem from one of the largest Portuguese carriers, which included generating feedback reports with indicators to send to the client, meeting with the client, take decisions based on the feedback, and the integration with the Field-Service system used by the client to present him our work planning proposal. To develop the fuel model, the solver used by the company was chosen which proved to be 60% faster than the other solvers for the passengers problem, while achieving better solutions. During the development of the fuel model, the feedback resulted in refining the model, providing in each iteration a solution closer to the client's needs. As the future direction of this work, new models will be developed and integrated into the system to deliver optimized work plans.

Keywords

Field-Service Management; Planning System; Route Optimization;

Resumo

Apesar do muitas empresas de serviço de campo planearem as suas tarefas manualmente, há software que ajuda as empresas na gestão e optimização dos seus recursos empregues em operações de serviço de campo. Na *Gestão e Otimização Operacional*, há procura por soluções adaptadas às necessidades do cliente, uma das soluções procuradas é a criação de planos de trabalho otimizados para logística e serviço de campo. Um plano de trabalho define um conjunto de tarefas a serem executadas por recursos humanos no campo, e pode ser avaliado com *Indicadores Chave de Desempenho*, permitindo assim medir a qualidade de um plano de trabalho enquanto comparado com outros planos.

No contexto deste estágio, a *Sentilant* desenvolve e vende um serviço de *Gestão e Otimização Operacional*. Um dos principais objetivos do serviço é gerar planos de trabalho para os seus clientes. Diferentes clientes da *Sentilant* têm necessidades específicas para os seus negócios, então há a necessidade de soluções à medida para esses e novos clientes. Para construir uma solução de planeamento à medida, a *Sentilant* iterativamente desenvolve um modelo com base no feedback do cliente, mas demora tempo a construir. A empresa utiliza um solucionador para implementar os modelos, mas procura alternativas para que possa reduzir o esforço temporal de desenvolvimento enquanto produzindo planos semelhantes ou melhores.

Para responder a este problema, o principal objetivo deste estágio é desenvolver um sistema de planeamento multi-inquilino que serve planos de trabalho optimizados contendo indicadores chave para permitir o cliente avaliar as suas soluções, e intervir na optimização para influenciar os resultados do plano.

Uma comparação extensiva de 3 solucionadores candidatos para desenvolver os modelos foi realizada, como a análise conceptual não foi suficiente para tirar conclusões um modelo foi implementado e comparado entre os solucionadores (modelo dos passageiros). Depois, utilizando componentes de software existentes na *Sentilant*, um sistema de planeamento foi desenvolvido para fornecer uma proposta de planos de trabalho optimizados para o problema do transportes de combustível de uma das maiores transportadores Portuguesas, o que incluiu gerar relatórios com indicadores para enviar ao cliente, reunir com o cliente, a tomada de decisões com base no feedback e a integração com o sistema de Gestão de Serviço de Campo utilizado pelo cliente, para apresentar-lhe a nossa proposta de planeamento. Para desenvolver o modelo dos combustíveis, o solucionador utilizado pela empresa mostrou ser 60% mais rápido do que os outros solucionadores para o problema dos passageiros, enquanto atingindo melhores soluções. Durante o desenvolvimento do modelo dos combustíveis, feedback resultou em refinar o modelo, entregando assim a cada iteração uma solução mais próxima das necessidades do cliente. Como direção futura deste trabalho, novos modelos serão desenvolvidos e integrados no sistema para entregar planos de trabalho otimizados.

Palavras-Chave

Gestão Operacional; Gestão e Otimização Operacional; Sistema de Planeamento; Optimização de Rotas;

Contents

Acronyms

API Application Programming Interface. v, 2, 3, 14, 20, 21, 29, 33, 36, 53, 59, 60, 64, 67

ASR Architecturally Significant Requirement. xv, 3, 23, 28, 29, 67

CC Cargo Center. xiii, 45, 48–50, 64, 68

CPU Central Processing Unit. 37, 54

CVRP Capacitated VRP. 6, 10–12

CVRPPDTW Capacitated VRP with Pickup and Deliveries and Time Windows. xv, 6, 8–11, 13, 21, 40, 46, 56

FSM Field-Service Management. 1–3, 7, 8, 19, 23, 29, 32, 33, 53, 62, 64, 67, 68

GLS Guided Local Search. 6, 12, 16, 54

GS Gas Station. xiii, 45–47, 49, 50

HTTP Hypertext Transfer Protocol. 33, 60

JSON JavaScript Object Notation. 60, 61

KPI Key Performance Indicator. xv, 1, 2, 7, 21, 23, 29, 36, 53–55, 58, 62, 67, 68

LA Late Acceptance in Hill-Climbing. 7, 16, 54

MCVRP Multi-Compartment VRP. 46

OSRM Open Source Routing Machine. 32, 35

PE Preferences Elicitation. 3, 44, 46–50, 62, 67

QA Quality Attribute. xiii, xv, 2, 3, 23–29, 31, 37, 67

R&R Ruin-and-Recreate. 7, 9, 54, 55, 59

RAM Random Access Memory. 37, 54, 60

REST Representational State Transfer. 32

TS Tabu Search. 6, 7, 12, 16

TSP Traveling Salesman Problem. 5, 10, 12

VRP Vehicle Routing Problem. 5, 6, 8–13, 15, 16, 67

VRPPD VRP with Pickup and Delivery. 6, 9, 11, 12, 15, 39, 66

VRPTW VRP with Time Windows. 6, 9, 10, 12, 43

List of Figures

List of Tables

Chapter 1

Introduction

Field-Service Management (FSM) is concerned with the management and optimization of resources employed outside of the company's properties to perform field-service tasks. Despite the large number of FSM platforms on the market, several clients value tailor-made solutions, a solution is developed align to the business needs, obtained through direct contact with client.

At a software level, an FSM system involves the optimization of the client's resources. For this purpose, optimization algorithms are used to find good approximated solutions for the client. When it comes to allocating human resources, vehicles, tasks and manage limited resources, an FSM system allows clients to obtain work plans. Those plans indicate a schedule of routes to guide the human resources in their field service jobs to successfully perform a set of tasks.

By involving optimization of resources, clients also value the quality of the plans, so besides routes, Key Performance Indicators (KPIs) are obtained as part of the work plans, allowing clients to take and improve field service decisions. Planned total distance to be traveled by a set of vehicles is an example of a KPI.

1.1 Context

This internship took place in *Sentilant*, a company that has been developing a system for FSM. As one of the components of the FSM system, *Sentilant* uses a routing optimization solver to build models and generate work plans, but stopped receiving major updates since 2016. Currently, *Sentilant* is **looking for alternative solvers to develop its models**. In addition to the need for an alternative, clients of the company are interested in work plans for field-service problems that *Sentilant* **currently has no answer** as the fuel problem, but also in better quality work plans for the passengers transport problem e.g., more tasks assigned.

In the **fuel problem**, the client wants work plans with assigned fuel orders (i.e., tasks) from gas stations to their drivers of tractors with cisterns. A cistern is composed by a set of isolated compartments. The load in each compartment must be optimized between the compartments to avoid destabilizing the tractor. For the **passengers problem**, the client wants a work plan for the transportation of people between accommodations and the airport, and vice-versa.

One of the FSM concerns is to ensure that the solution provided meets the client's needs,

since their logistic decisions and resource management highly depends on a set of Quality Attribute (QA) of the software systems they are using. On the client-side, there is a **high concern with the quality of the work plans**. Some clients want to intervene in the optimization to change the quality of their plans. For example, requesting the possibility of their employees being able to work longer, allocating more tasks, that otherwise would not be assigned.

As the company responds to new clients within the same solver, the source code becomes more difficult to maintain. Modifications to the source code can unintentionally change the behavior of the models that were already validated by their respective clients. Thus, in parallel with the client's needs, the company values a system that uses evolving i.e., frequently updated technologies prepared for continued update and ease the integration of new models. By offering the **ease of developing and adding new models**, the *Sentilant*'s team can **reduce the cost of developing a model**.

1.2 Objectives

The main objective of this work is to contribute to the enrichment of the FSM system of the company, by developing software components and offering a set of quality attributes aligned with the needs of *Sentilant* and their clients in a mid-long term. To achieve this main objective, we divide into the following goals:

- **Develop and deploy a multi-tenant planning system delivered as an API, allowing clients to obtain optimized work plans:** To achieve this goal, we did requirement gathering centered on QAs, derived an architecture for the system and implemented the system. To derive the architecture, a solver technology was chosen to develop a fuel model and integrate into the system as the first tenant, thus serving this client with work plans;

- **Offer optimized work plans to answer real clients problems as depicted in 1.1:** To achieve this goal, an extensive analysis to solver technologies was performed to find the best solver that can meet the client problem requirements, while complying with the QAs;

- **Guarantee to clients a way to evaluate their solutions by delivering a set of KPIs:** Based on the result from the optimization, metrics are extracted and appended to the plan received by the client. For this, the KPIs for the passengers and the fuel transport problem were identified;

- **Offer clients an interface to parameterize the model (Quality of the Solution and Performance):** For instance, in the case of the passengers problem the client wants to reduce the distance traveled in empty by the drivers, or in the fuel problem configure the loading time of the cistern. Therefore, the parameters that the client wants to configure were identified and added to the model as configurable. This is, in addition to the problem's data a model accepts additional parameters as input;

- **Integrate the API in the FSM service** used by the fuel problem client to close a development cycle of a tailored planning solution for a tenant in this work. The integrated solution serves as demonstration to the client.

1.3 Document Structure

Chapter 2 reviews technical knowledge necessary to understand the root of our plans and how they are computed: the main types of routing problems and meta-heuristic algorithms. Then, an extensive analysis and comparison between solvers. In **Chapter 3** is explained the software development process, planning versus work executed, and risks identification. **Chapter 4** depicts the architectural drivers (ASR) of the project: high-level overview of the functional requirements, QAs, and business and technical constraints. **Chapter 5** defines the architecture of the system and the developed components. **Chapters 6** defines the problems and documents the models in mathematical notation for the passengers and fuel problems, respectively. For the fuel model, Preferences Elicitation (PE) are given to justify the implementation decisions, based on feedback from the client. **Chapter 7** presents results: comparison between two solvers configured in this work and the *Sentilant*'s solver for the passengers problem. Data showed the *Sentilant*'s model found better quality solutions in less time. Therefore, *Sentilant*'s model was extended to answer the fuel model and results presented. Also, results regarding the Planning System API, the feedback report presented to the client and the integration with the FSM service is presented. Finally, **Chapter 8** summaries the work performed and proposes future work.

Chapter 2

Background Knowledge

The key component of this system, a solver, uses an optimization algorithm to find a good solution given a problem instance from a client. Since those problems involve routing, in 2.1 is explained how vehicle routing is defined in the literature and how it is usually solved. To contextualize the reader, in 2.2 is described the work plan and what information should be included in it. Section 2.3 details the identified tools candidates to generate optimized plans. Then, a comparison of the tools was performed in 2.4 and conclusions took in 2.5.

2.1 Optimization

The work plans to be generated involves routes, in 2.1.1 is explained the main type of routing problems and introduce to concepts in the literature. Section 2.1.2 describes algorithms used in this work to answer this optimization problem. Then, section 2.1.3 explains what is a solver in our context, the over-constraining scenario, and termination criteria.

2.1.1 Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is a combinatorial optimization problem that aims to find an optimal set of routes. Giving a formal definition from the book, "Vehicle Routing Problems: Methods, and Applications", Toth et al. (2014) defines a VRP by "**Given:** A set of transportation requests and a fleet of vehicles", the task is to define a plan for "**Task:** Determine a set of vehicle routes to perform all (or some) transportation requests with the given vehicle fleet at minimum cost; in particular, decide which vehicle handles which requests in which sequence so that all vehicle routes can be feasibly executed".

A VRP can be divided into different variants, the simplest variant is the Traveling Salesman Problem (TSP). Given a set of n cities, a salesman has to visit each one of the cities, taking the most efficient route that minimizes the distance traveled by the salesman. An exhaustive search of all possible paths guarantees to find the shortest path, but it's computationally intractable, the search space grows exponentially $n!$ and becomes time-expensive to find an optimal solution. TSP is a well known NP-hard problem in the literature reasonably solved with combinatorial optimization techniques. TSP and the main VRP variants are usually formulated as discrete problems solved with combinatorial optimization techniques that iteratively seeks to find improving solutions from the finite set of possibilities i.e., search space.

Depending on the authors describing the VRPs variants, they are described differently.

One of the most influential books on VRP: "The Vehicle Routing Problem" by Toth and Vigo (2002) defines the four main types of VRP as particular variants of it:

Capacitated VRP (CVRP) where all vehicles are identical and based at a single central depot, only capacity constraints for the vehicles are imposed. The goal is to minimize the total cost (i.e., distance or travel time) to serve all the customers;

VRP with Time Windows (VRPTW) is an extension of the CVRP in which capacity constraints are imposed and each customer is associated with a time window. Each customer must be served between the time windows $[a_i, b_i]$, and the vehicle must stop at the customer i location during a s_i time instants within the time interval. The goal is to find a collection of exactly K simple routes with the minimum cost, such that: i) each route visits the depot, ii) each customer is visited by exactly one route, iii) the sum of the demands of the customers visited by a route does not exceed the vehicle capacity, C and iv) for each customer i, the service starts within the time window $[a_i, b_i]$ and the vehicle stops for s_i time instants;

VRP with Backhauls (VRPB) is an extension of CVRP in which the customer set is partitioned in two subsets. The first subset contains Linehaul customers each requiring a given quantity of products to be delivered. The second contains Backhaul customers, where a given quantity of inbound products must be picked up. Whether a route serves both types of customers, all linehaul customers must be served before any backhaul customer;

VRP with Pickup and Delivery (VRPPD), each customer i is associated with two quantities d_i and p_i, the demand to be delivered and picked up at customer i, respectively. In some cases, only one demand quantity is used $d_i = d_i - p_i$ for each customer i, indicating the net difference between the delivery and the pickup demands. For each customer i, O_i denotes the origin of the delivery demand and D_i the destination of the pickup demand. Find a collection of exactly K simple routes with minimum cost, such that: i) each route visits the depot; ii) each customer is visited by exactly one route; iii) the current load of the vehicle along the route must be non-negative and must not exceed the vehicle capacity C, iii) for each customer i, the customer O_i, when different from the depot, must be served in the same route and before customer i, and v) for each customer i, the customer D_i, when different from the depot, must be served in the same route and after customer i.

The models developed in this work are **Capacitated VRP with Pickup and Deliveries and Time Windowss (CVRPPDTWs)** that involves properties from three of the main types of VRP: CVRP, VRPTW and VRPPD.

2.1.2 Meta-heuristics

As opposed to exact methods, which guarantee to give an optimum solution to the problem, heuristic methods attempt to find good solutions within a realistic time frame but not necessarily optimum solutions. The algorithms that showed better results by the solvers in this work are as follows, better is defined by assigning more tasks giving a similar running time. These algorithms belong to the family of local search algorithms that move from solution to solution in the search space by applying local changes.

Tabu Search (TS) where the name tabu comes from the fact that some moves are temporarily declared as forbidden (i.e., tabu), this restricts the move selection. Allows moves that deteriorate the current objective function value;

Guided Local Search (GLS) focus on exploitation of the problem and search-related information to guide local search in the search space. This is achieved by augmenting

the objective function of the problem with a set of penalty terms that are dynamically manipulated during the search process to steer the heuristic, Voudouris (1998);

Late Acceptance in Hill-Climbing (LA) first published by Burke and Bykov (2008), compares the candidate solution with a solution, which was the "current" solution several steps before, opposing to Hill Climbing that compares a candidate solution with the current;

Ruin-and-Recreate (R&R): A meta-heuristic developed by Schrimpf et al. (2000) that combines elements of simulated annealing and threshold-accepting algorithms with bold, larges moves instead of smaller i.e., against the local search idea of applying a local change to move to a neighbor solution. Every time during the R&R optimization there is always an admissible solution, in the ruin step parts (e.g., tasks) are disintegrated from the solution and in the recreate step the solution is rebuilt.

Our intention is to use **tools that facilitate the creation of models for FSM**, presenting an optimization algorithm suitable to solve routing problems as a basis for the development of our models.

2.1.3 Additional Concepts

A **solver** is a mathematical software used to model and/or solve mathematical problems. Distributed as a stand-alone program, as a software library or even as a complete framework. The most known are general-purpose to solve linear, integer, and constraint satisfaction problems. Our problems involves routing optimization, therefore choosing the solver is one of the most important decisions in this work, since we are confined to its functionalities and performance. Under the context of optimization modeling in a solver there are **two types of constraints**: i) **hard constraint** that must be satisfied by any **feasible solution**. E.g., a driver cannot drive more than 7 hours in a day, and ii) **soft constraint** that can be violated, but has an associated **cost/penalty** added to the **objective function**. E.g., penalize the distance traveled with a vehicle in empty.

In some problems is convenient letting solutions be feasible without assigning all objects e.g., tasks, this is known as **over-constraining problem**.

Without setting a **termination criteria** the local search algorithm runs forever, consequently never stops and returns a solution. From Gendreau (2006), the commonly used termination criteria in TS (also applied to the remaining meta-heuristics), stop: i) after a fixed number of iterations without finding a better solution, ii) after a fixed number of iterations or processing time, and iii) when the solution reaches a defined threshold value.

2.2 Work Plan

The work plan is the information that clients are interested in. Our objective is to obtain a balance between performance and solution quality of the optimization that satisfies the client. The work plan consists of a set of routes, each route is made up of a set of tasks to be performed sequentially. The plan has aggregated KPIs i.e., metrics, for the plan as a whole, and for each route. Figure 2.1 illustrates a generic plan.

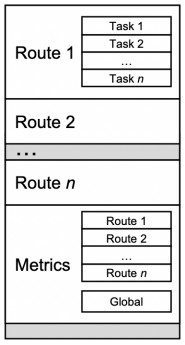


Figure 2.1: Illustration of a work plan

Metrics do not strictly depend on the type of field service activity of the client, sometimes **clients have personal preferences**. However, there are mandatory metrics that are provided in a work plan.

The following metrics are included to each route, global to the plan e.g., sum of each route, and averages.

- **Tasks assigned:** The number of tasks assigned, given the overconstraining cases when all tasks could not be assigned;
- **Distance**
  - Total distance: Distance traveled between locations;
  - In empty: Distance traveled between location when the vehicle is empty;
- **Duration**
  - Driving: Driving duration between locations;
  - Service: Duration to perform each task on the place;
  - Effective: Driving and service duration together;
  - Route: Duration between the actual start and the end of the route;
  - Work: Duration between the start of a work block and the end of the route. The work duration is greater than or equal to the route duration;
  - Wait: A driver can arrive at a location earlier, so there is a waiting time;
- **Cost**
  - Resources: The cost of the resources (e.g., drivers), usually per hour;
  - Vehicles: The cost of the vehicles, usually per kilometer plus the fixed costs;
  - Work: The resources cost plus the vehicles cost;
- **Average speed:** The average speed of the vehicles;
- **Monetary gain:** The monetary gain of the assigned tasks.

2.3 Technological Choices

The solution employed by *Sentilant* uses *JSprit* solver for developing the models, and providing work plans as a part of the FSM system. Currently, there are many tools on the market labeled as VRP, however, only a portion of them answer the **CVRPPDTW** as *JSprit* does. Some are targeted to routing optimization such as *JSprit*, yet others are generic to solve various combinatorial problems, letting extend with custom functionality.

During the **first step**, documentation and examples were analyzed to understand the potential of each tool, thus any tool that provided much fewer functionalities than *JSprit* or didn't allow CVRPPDTW models was excluded. For each tool in sections 2.3.1-2.3.3, a summary of examples, documentation, and some relevant information about the optimization algorithms and constraints, as this is the core of the illustrated tools. Then, the main limitations of the tools are presented.

As a **second step**, meta-information about the tools was gathered, such as frequency of version update, relevance on GitHub, StackOverflow, and main uses-cases, trying to create a large distinction between them. Technical aspects were also compared, assuming *JSprit* would not be used in this work. The results of the analysis are depicted in the next section, although before providing the comparison the reader should form an idea about each solver.

In **Optimization Algorithms and Constraints**, a summary of the supported constructive heuristics, meta-heuristics and contextualizes how hard and soft constraints are implemented. **Documentation and Examples** summaries the conditions of the documentation and VRP examples provided by the tool. **Selection and Custom Moves** only applies to *OptaPlanner*, explaining how can be improved the performance for problems with pickup and deliveries. Then, **limitations** of the tools depicting the main downsides of each tool to implement our planning problems.

2.3.1 Technology 1: JSprit

JSprit (*jsprit.github.io*) is an open-source tool for solving VRPs. It can solve a wide range of VRP problems such as VRPTW, VRPPD, heterogeneous fleet, multiple depots, and with distinct start and end locations. Unlike the technologies in 2.3.2 and 2.3.3, *JSprit* is not supported by a large company like *RedHat* and *Google*, respectively.

Optimization Algorithms and Constraints: Uses the best insertion as construction heuristic and offers variants of R&R. The R&R is extended with strategies based on the work of Pisinger and Ropke (2007). The algorithm is suited to solve complex problems that have many constraints and a discontinue solution space. Can be extended with a custom cost function, hard and soft constraints.

In the R&R an iteration of the algorithm involves a ruin and a recreate step. In the ruin step giving a set of routes and assigned tasks, a random number of tasks are disintegrated from the solution and put into a bag. In the recreate step a best insertion is followed. The hard-constraints are evaluated before any insertion, and a task is not inserted if it breaks any of the constraints, thus a feasible solution is always obtained at the end of every iteration. Some strategies can be selected in *JSprit*. For instance, in the radial ruin given a random location v and a random probability r, the r nearest locations from v, including v, are disintegrated from the solution.

Documentation and Examples: Provides source code of 40 executable examples, and it seems straightforward to solve the main VRP variants. Moreover, it provides a graph viewer to easily visualize the various routes obtained from the best solution found. It does not contain a specification document, just a set of text files describing some of the examples and an overview of the main components. Therefore, the way to learn how to interact with this tool is to follow the examples and read the source code. A CVRPPDTW solution is already implemented, and it follow a builder pattern to set them up.

Limitations: Restricted to already implemented VRP features such as settings up the vehicles, pickup and deliveries, services, and time windows. The service is done on the place, whereas a shipment has a respective pickup and delivery place. In detail, the shipment object allows setting location, service time, various time windows of both pickup and delivery, dimension and capacity, various required skills to perform the task, maximum time on the vehicle, and priority of the task. However, complying with custom functionalities proves to be a more complex task, but this limitation also proves to be an advantage, since it guides the developers to quickly implement a solution to solve the main VRP variants.

To implement a custom hard-constraint, it requires to implement the logic whether an insertion will lead to a feasible or infeasible solution as part of the recreate process of the R&R. However, in the meta-heuristics used by 2.3.2 and 2.3.3 technologies, first the move is performed, then the hard constraints are simply evaluated.

2.3.2 Technology 2: OptaPlanner

OptaPlanner (*optaplanner.org*) is self-claimed as constraint satisfaction framework, which optimizes planning problems through constraint satisfaction programming targeted to aid organizations to provide products and services with a limited set of resources. Self-claimed as stable, reliable and scalable, used in production throughout the world.

The solver is not provided as a "pre-made model" to solve VRP problems, however it provides examples of TSP and CVRP. A CVRPPDTW can be modeled, but there are performance gaps as the meta-heuristics and local search moves are not built in a dedicated way for routing problems. To understand the wide scope of *OptaPlanner*, it provides a set of use-cases to solve VRPs, employee rostering, maintenance scheduling, task assignment optimization, school timetabling, cloud optimization, and conference scheduling.

Optimization Algorithms and Constraints: A problem is modeled with Java classes to define the business domain model and implement the required logic. Lets us define the data (i.e., facts of the problem) and supports several decision variables (i.e., planning objects). After this step, constraints of the problem can be described. Provides 3 main ways to implement constraints: i) easy score, ii) incremental java, iii) drools. The first one is straightforward, yet it is slower than the others, since it calculates the entire cost on every solution evaluation. It supports multi-threading for incremental java and drools with significant performance gains. It's advised to use drools as the second is time-consuming and difficult to implement. Drools is a business rule management system with a forward-chaining and backward-chaining inference, based rules engine that reacts to changes in data and provides built-in temporal reasoning, Proctor (2012). Most of the Drools syntax uses forward chaining and does incremental calculation without extra code.

Supports 8 constructive heuristics, for VRP first fit decreasing is advised, several meta-heuristics: hill climbing, tabu search, simulated annealing, late acceptance, great deluge, step counting hill-climbing, and variable neighborhood descent meta-heuristics, and also exhaustive search i.e., brute force algorithms such as branch and bound.

Documentation and Examples: The framework provides a well-documented specification (*docs.optaplanner.org*). It provides 2 examples of VRPs and encourages users to adapt the examples to solve their business problems. The first example is the tsp, the second is a CVRP with the option to use time windows VRPTW. These examples are important to understand which mechanisms *OptaPlanner* provides to implement VRPs. Apart from the technical details, the specification also details each of the examples. It should be noted that every example provided with *OptaPlanner* represents a real optimization problem. The examples are not large datasets, nevertheless, they may represent real case scenarios.

Selection and Custom Moves: A complex functionality is the ability to design a custom behavior to the local search moves. To have control over moves they are structured in a tree as depicted in 2.2, the root move selector creates an iterator of moves to be injected in the optimization algorithm in every iteration. The *unionMoveSelector* selects a *Move* by selecting one of the *MoveSelector* children (in the Figure are *changeMoveSelector* and *swapMovelSelector*). A set of generic moves are supplied such as change move that changes 1 entity's variable value and swap that swaps variables of 2 selected entities.

According to the creator of this tool, design a custom move is complex, and usually leads to conflicts with other mechanisms of the framework, besides, the debugging is usually complicated. This is because there is a set of behaviors that the framework takes that are not trivial to understand and no documentation is provided in this regard. A custom move can involve change, add and remove of several decision variables, in a single atomic

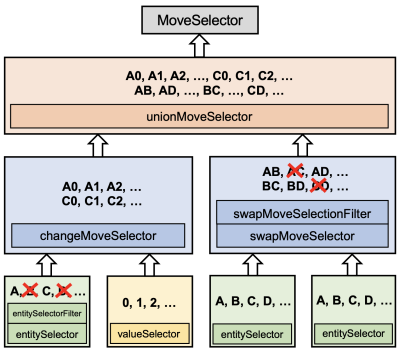


Figure 2.2: Example of the structure of the selector tree in the *OptaPlanner*'s architecture to generate the next move of the local search (Adapted from *Optaplanner*'s specification).

move. Moreover, non-doable moves can be filtered that: 1) changes nothing, 2) impossible to perform in the current solution. When optimizing with multi-threading, additional behavior has to be implemented to migrate moves from one thread to another. Furthermore, custom moves can be generated, for example, in scenarios with pickup and delivery, 2 pairs of entities to be changed or swapped in just one move can be generated.

Limitations: The VRPs are not abstracted, the logic has to be written, implementing problems such as VRPPD with hard constraints provides bad results, the move is constantly breaking the constraints and the local search is inefficient, based on the *OctaPlanner*'s founder Geoffrey De Smet. Based on our searches, VRPPD cannot be easily solved with *OptaPlanner*, even if solved, the logic is prone to error and causes performance issues.

The straightforward implementation of enforcing pickup and delivery through constraints yield bad performance results. Workaround: Custom moves: move pickup and delivery together, constraining other functionalities (From: Redheat issues PLANNER-833).

For overconstraining, it uses a chaining variables feature to generate a chain of visits that ends on an anchor e.g., start or end location of the route. This feature does not support the unassignment of variables, thus hindering the performance of the algorithm. Workaround: Assign the dropped visits to a ghost vehicle (From: Redheat issues PLANNER-226).

As a complement, a specialist in meta-heuristics Gendreau (2006) says that in local search: "the quality of the solution obtained and computing times are usually highly dependent on the “richness” of the set of transformations (moves) considered at each iteration".

2.3.3 Technology 3: OR-Tools

Google Operational Research Tools (*OR-Tools*) is an open-source software tool for solving optimization problems and it's supported for Python, Java, C# and C++. One of the provided solvers is a Routing Library (RL) for solving node-based problems such as VRPs, and also arc-based problems. Implemented as a "single model" to solve a wide range of routing problems, such as CVRPPDTW. Google uses *OR-Tools* to plan the shortest routes for Google Street View cars. *OR-Tools* provides modules to solve bin packing, network flows, employee scheduling, and the job shop. As an operation research tool, it offers linear, constraint, and integer optimization solvers.

As a fact to prove the *OR-Tools* performance, Surana (2019) performed a benchmark to measure the performance of *OR-Tools* to solve CVRP problems from the Networking and Emerging Optimization organization. It provides 3 benchmark sets with different criteria

that must be met, such as the maximum running time of the solver. The authors ran all combinations of constructive heuristics and meta-heuristics for each instance and concluded that almost 60% of the best solutions for each CVRP found better results than the current best-known from the *Spain Networking and Emerging Research Group* benchmark.

Optimization Algorithms and Constraints: Model is configured in a procedural programming pattern. To implement custom hard constraints, a single line notation is used, for instance, *routing.VehicleVar(pickup_index) == routing.VehicleVar(delivery_index)* telling that nodes *pickup_index* and *delivery_index* have to be served by the same vehicle. Regarding soft constraints, the solver only supports: i) transition cost between nodes, ii) lower and upper bounds that penalize whether the solution exceeds defined cumulative bounds, iii) set nodes as optional to be visited, whether a node is dropped penalize with a certain cost, iv) fixed cost for using the vehicle. First runs a constructive heuristic to find an initial solution, and then a meta-heuristic, offering 10 constructive heuristics e.g., local cheapest insertion and 5 meta-heuristics e.g., TS and GLS.

Documentation and Examples: The documentation of the RL (*developers.google.com/optimization*) is a guide with brief explanations accompanied by source code. Those examples explain how to add capacities, time windows, pickup and deliveries, and introduce to the dimension mechanism to configure cumulative variables. Each example is dedicated to explain a functionality apart from real use cases, unlike *OptaPlanner* which focuses on explaining functionalities from a real use case example. As a downside, there is no formal documentation that details neither the *OR-Tools* nor the RL.

Limitations: Implemented as "one single model", so its pattern has to be followed, like *JSprit*. The way the model is coded the driver only starts and/or arrives at the depot, only supports a fixed transition cost between nodes for each vehicle that cannot be influenced by other variables during optimization. A custom soft constraint only can access one variable (e.g., distance) for penalization. For instance, to penalize the distance in empty traveled by a vehicle, it's necessary to acess capacity to know that the vehicle is empty, this is impossible in *OR-Tools*. When transporting people in a taxi-mode pickup and delivery scenario, before the first pickup and after each delivery to the next location the vehicle is always empty, so we can set this type of penalization in *OR-Tools*. However, when dealing with mixed pickup and deliveries is never know when the vehicles are empty, there is a dependency on the vehicle's capacity to add the penalty. In the C++ version of *OR-Tools* there are two mechanisms called *State Dependent Transit* and *State Dependent Dimensions* that allows to access different variables to lead with these problems, but the feature is experimental and currently does not work well.

2.3.4 Other Technologies

According to our searches, *JSprit*, *OptaPlanner* and *OR-Tools* are the main tools to answers VRP in a planning point of view where various constraints are considered. In any case, a list of other tools that could be candidates is presented: i) **VRPH**: An open-source C++ library (*projects.coin-or.org/VRPH*) for solving VRPs, developed by Chris Groër in 2010. In some cases, it can find solutions that are within about one percent of the best-known solution on benchmark problems Groër et al. (2010), ii) **VROOM:** An open-source optimization engine (*vroom-project.org*) written in C++ to solve various real-life VRP; and iii) **Open-VRP:** A Lisp library (*github.com/mck-/Open-VRP*) to solve TSP, VRP, CVRP, VRPTW, and VRPPD released by Marc Kuo in 2013.

2.4 Comparison Vectors

Although a CVRPPDTW can be answered with any of these tools, these 3 tools have strengths only detected when compared side by side. Therefore, two types of information were collected: i) meta-information about the tools e.g., license and ii) technical information. Moreover, as a requirement of the company, only free tools were considered.

Assuming that more than one tool can answer our problem, the **meta-information** is more relevant than the technical characteristics of the tool for the company in a mid-long term. The following parameters were collected:

- **Free:** Is this software library free or not?;
- **Paid Features:** Does it offer any supplementary paid service(s)?;
- **Features:** If paid services are offered, what services are these e.g., technical support?;
- **License:** While there are open source tools, some have licenses that do not allow us to use the tool for free. What license does it have?;
- **Documentation:** Without documentation the way to understand a tool is to reverse engineering it. Does the tool present any kind of meaningful documentation?;
- **Documentation Specification:** Does the tool present a formal document that details the architecture and how to interact with the tool;
- **Code Examples:** A solver without examples of use is very suspicious in terms of its quality. How many usage examples does it provide?;
- **Working Examples:** From the examples provided by the tool, how many did work on our personal computer?;
- **Specialized for Routing:** Is the tool specialized for routing optimization or can it also be used to solve other optimization problems?;
- **VRP Code Examples:** How many examples does it provide to solve VRPs?;
- **Languages:** What language does the tool support? This point is important as it may not be supported in a language adopted in the company. They are Python and Java;
- **Descendant Projects:** Do the developers of the tool have any project descending from their tool?;
- **Easy Install:** Does the tool have a simple installation on different operating systems (e.g., MacOS, CentOS, Windows) or does it involve dependencies that are difficult to install or outdated?;
- **Use-Case:** What is the usage scenario that the tool is aimed at?;
- **Production Ready:** Can we use this tool in a production environment? This is a common question addressed to the developers of the tools and it is usually possible to get an answer. It may work to obtain a few plans, but may not be prepared to run in a server environment constantly processing requests;
- **GitHub Starts:** Giving a star to a GitHub (*github.com*) repository means that the follower will receive notification of updates and will be able to quickly find the repository. The number of stars is related with the number of releases Borges et al. (2017);

- **Google Trends:** (trends.google.com) ranks search terms as trends with an interest over time. Tools are sorted by the interest rate of the last 5 years, a lower value means more interest rate;
- **StackOverflow Questions:** The StackOverflow is the largest question answering website about software, the website domain is in 39th place on the top sites *onalexa.com* (26th of February 2020). How many questions are posted in the StackOverflow that tags this tool;
- **Software Type:** At the developer level, how is it delivered? Usually in the form of a software library, but it can be a framework that provides a development environment;
- **Releases:** Initial release, current version, last release, previous version, previous release, update frequency: This information is extremely important for the company. A tool that stays up to date and evolves over time is one of the strongest points for choosing a tool, considering that it can functionally answer our problems;
- **Company:** Which company owns the project? Being a large company, more guarantees are given that the quality of the tool is ensured, comparing to a tool, for instance, maintained by a doctoral student.

Table 2.1: Comparison of the meta-information about the tools on 27th February 2020

	JSprit	**OptaPlanner**	**OR-Tools**
Free	Yes	Yes	Yes
Paid Features	Yes	Yes	No
Features	Route Optim. API	Tech Support	None
License	Apache v2	Apache v2	Apache v2
Documentation	No	Yes	Yes
Doc. Specification	No	Yes	No
Code Examples	40	22	93
Working Examples	All	All	All
Specialized for Routing	Yes	No	Yes (RL)
Routing Code Examples	40	3	16
Languages	Java	Java	C++, Python, C, or Java
Descendant Projects	Route Optim. API	OptaWeb	–
Easy Install	Yes	Yes	Yes
Use Case	VRP	Planning Problems	Combinatorial Optim.
Production Ready	Yes	Yes	Yes
GitHub Stars	982	1477	5002
Google Trends	3	1	2
StackOverflow	84	956	348
Software type	Library	Framework	Library
Initial Release	2013	2004	2015 (As open-source)
Current Version	v1.8	7.33.0.Final	v7.5
Last Release	Apr 10 2019	Feb 17 2020	Jan 28 2020
Previous Version	v1.7.2	7.32.0.Final	v7.4
Previous Release	June 8 2017	Jan 28 2020	Oct 11 2019
Update Frequency	Low	Very High	High
Company	GraphHopper	RedHat	Google

Table 2.1 depicts the results of the meta-information analysis. The Route Optimization API is a paid service that uses *JSprit* to obtain optimized routes since its launch *JSprit* stopped receiving major updates. The *OptaWeb* is a set of free projects that uses a Web GUI to optimize planning problem, using the *OptaPlanner* solver.

While trying to implement the passengers model in the *OR-Tools* and *OptaPlanner*, technical restrictions of each tool were identified. The **technical points** are as follows:

- **Model:** Not all solvers allow to design the problem model. *OptaPlanner* allows to model the problem domain, but cannot compete with the performance of *OR-Tools* and *JSprit* as a "pre-made model" targeted to specifically solve complex variants of the VRP;
- **Cost function** evaluates the quality of the solutions found. *OptaPlanner* offers 5 different ways to write constraints in each one the performance differs, whereas *OR-Tools* is attached to an objective function in which only the penalty values can be changed, and *JSprit* supports two types of soft constraints: cost function and insertion costs;
- **Custom Soft Constraints:** For example, to maximize the number of tasks performed and minimize drivers' working hours. All solvers support these soft constraints in the cost function, so it can be consider to support custom soft constraints;
- **Multi-Variable Constraint:** In a given objective, we want to minimize the empty distance traveled by vehicles. For this penalty, it is necessary to know if the vehicle is empty to be able to add a penalty cost with a value proportional to the distance. That is, there is a dependency in the value of the capacity of the vehicle; *OR-Tools* does not allow this type of constraint, only to define upper and lower bounds for a single variable, if the value is exceeded than add a penalty to the cost function;
- **Hard Constraints** are supported by all solvers, however, the implementation method in *OR-Tools* is very counter-productive, since each condition has to be added individually. This mode is not maintainable. In the case of *OptaPlanner*, the constraints can be described in Drools which makes the constraints highly understandable, whereas in *JSprit* are implemented in traditional Java programming;
- **Cumulative Variables:** Vehicle load, time, distance traveled (among others) are accumulated along the route. There are arrival and departure times for each task that the solver should calculate during optimization. *OR-Tools* introduces the concept of dimension to configure a cumulative variable. *OptaPlanner* provides listeners who, according to the moves, recalculate the value of those variables, whereas *JSprit* let be extended with custom states similar to *OptaPlanner*'s listeners;
- **Algorithms Configuration:** *OptaPlanner* and *JSprit* allows to change configuration parameters of the algorithm, for *OptaPlanner* it also allows defining how certain construction heuristics build the initial solution. In *OR-Tools* construction heuristic and the meta-heuristic are simply selected to be used;
- **Termination Criteria:** All tools allows to define the termination criteria of the meta-heuristic. It may involve several conditions such as unimprovement time elapsed, unimprovement iterations elapsed, time limit and custom implementations;
- **Custom Moves:** Only *OptaPlanner* allows to change the selection and behavior of the local search moves. In terms of routing, *OptaPlanner* has not implemented an optimized way to respond to the VRPPD. The only way is resorting to custom moves to avoid breaking constraints. In the case of *OR-Tools*, a method is provided to indicate pairs of pickup and delivery where internally it uses a richer way to move them. *JSprit* also has a dedicated implementation, although it cannot be extended with custom functionality.

For the best of our knowledge, a comparison between these tools was not found at this level. A summary of the comparison of technical details of the tools is depicted in 2.2. Design a model can create ambiguity for the reader. In the company, a model is an implementation that responds to a client's problem. This analysis refers what solvers offer to implement a model. For example, *OR-Tools* is confined to the solver's architecture that is provided in the form of a model, in which the decision variables cannot be changed.

Table 2.2: Comparison of technical details between *JSprit*, *OR-Tools* and *OptaPlanner*. The checkmark means the tool has the functionality.

	JSprit	OptaPlanner	OR-Tools
Design a Model		✓	
Change Cost Function	✓	✓	
Custom Soft Constraints	✓	✓	✓
Multi-Variable Soft Constraint	✓	✓	
Custom Hard Constraints	✓	✓	✓
Cumulative Variables	✓	✓	✓
Algorithms Configuration	✓	✓	
Termination Criteria	✓	✓	✓
Custom Moves		✓	

In line with the technical analysis, there are 3 subjective points to highlight. In terms of **performance**, for single-objective VRP variants, there are few tools that can compete with *OR-Tools*' performance. *OptaPlanner* and *JSprit* support multi-threading which can boost the number of evaluated solutions per seconds and find a solution faster, yet we are interested in a balance between quality and performance. For **usage simplicity**, the *OptaPlanner* is the most complex in terms of features. The nomenclature of the methods used by *OR-Tools* is not pleasant and it is not easy to understand, *JSprit* is in the middle it is the most readable, but provides the worst documentation.

More important than the **meta-heuristic** being use is the quality of the solutions found and the performance of the algorithm. Since these algorithms are complex, their implementation differs from tool to tool even for the same algorithm. In *OR-Tools* the GLS is the best in most cases and in *OptaPlanner* for our problem that involves routing is LA. Making a fairer comparison with the same algorithm, the TS implementation in *OR-Tools* showed better results in the passengers problem than the TS in *OptaPlanner*.

2.5 Discussion

Our field-service problems involves routing in its core. The VRP is one of the most important problems in the combinatorial optimization literature with applicability in the industry. These solvers can be adjusted, allowing us to obtain optimized work plans, **without care about the underlying implementation of the optimization algorithms**. Our effort should be directed towards meeting the client's needs.

As a planning tool, the *OptaPlanner* allows us to design models for a variety of combinatorial problems. There is a loss in performance related to dedicated routing tools since those dedicated tools use specific implementations select on purpose to solve variants of VRP. The *JSprit* and *OR-Tools* provides a straightforward way to implement the most common VRPs, however, they lack extensibility to change the behavior of particular components of the solver as *OptaPlanner* that can be used in our advantage.

By seeking a balance between performance and quality of the solutions provided by the solver, the solution quality of the work plans highly depends on the performance of the solver to find better solutions in the shortest time possible. A study that compares these solvers was not found to help us make a decision. As the **solvers showed very different strengths**, they were explored with the passengers problem. Comparing results obtained from the models implemented in these tools to solve the same problem, is an asset for *Sentilant*.

Chapter 3

Planning Process

The followed software development process, involved parts, and the communication mechanisms used to communicate with the team in section 3.1. The work planned versus performed in 3.2, then in 3.3 the risks derived from the thresholds of success.

3.1 Development Methodology

In software engineering one of the most important steps is the definition of the development methodology. *Sentilant* uses the *Agile* development process for the development of its projects. There is no confinement to this methodology and given that *Sentilant* is a company with a reduced number of employees in which much of the information is transmitted verbally, an *Agile* development approach is followed.

The *Agile* development process allows to focus on responding to changes rather than following a rigid development plan. There is still a plan, but it gives flexibility to accommodate changes. Despite being only one system, the solution for each client is tailored according to his needs. These needs are dynamic, with *Agile* the system benefits from incrementally improving the system to answer new clients and modifying the existing models according to the field-service requirements of the clients. It also allows to prioritize requirements, since requirements can be postponed to future iterations, when better information or structure is available, thus optimizing the efficiency of the client's choices.

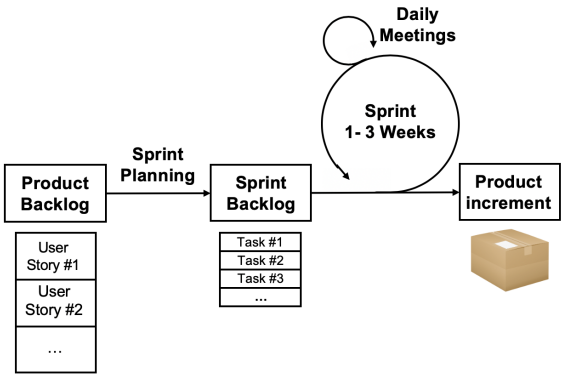


Figure 3.1: Agile development process

In Figure 3.1 is depicted the development process (inspired on *Scrum*). In summary, the beginning of the development of the project starts with a *Sprint* where feature(s) are

selected to be implemented during the *Sprint Planning*, the end of the *Sprint* corresponds to an increment to the product.

The *Product Backlog* corresponds to a list of user stories wished to implement in the product, ordered by priority. The *Sprint Backlog* is a set of tasks derived from the *Product Backlog* committed to implement in the product until the end of the *Sprint*. This choice is based on priority and developers' perception of how long it will take to implement each item discussed in the *Sprint Planning*. The *Sprint Planning* is a meeting held before start the actual *Sprint*. During the course of a *Sprint*, as part of the *Daily Meetings*, a meeting is held with the team to report the progress and find possible solutions for problems that have occurred or may occur. After the end of the *Sprint*, the product is incremented and a the next iteration is discussed in the *Sprint Planning* meeting.

The company uses the *Slack* platform to establish communication between the team. For the meetings, *Zoom* platform is used. Moreover, the advisor from *Sentilant*, Eng. Bruno Cabral represents the client for the passengers problem and for the fuel transports problem the client is one of the largest Portuguese carriers.

3.2 Planning

This section documents the tasks planned for both semesters and the actual work done. The work developed in the first semester was the review of the methodologies and technologies, necessary knowledge about the problem domain, implementation of the passengers problem to demonstrate the practical capabilities of the solvers, capture of architectural drivers and derive a preliminary architecture. For the second semester, the solver from *Sentilant*, that uses *JSprit*, was extended to solve the fuel transports problem, progress reports were generated and delivered to the client, and the API built and integrated for this tenant.

3.2.1 First Semester

In the first semester, the plan exposed in the internship proposal was followed. The work done and the planning is depicted in Figure 3.2. The following textual description helps the reader to interpret the plan and understand what was done.

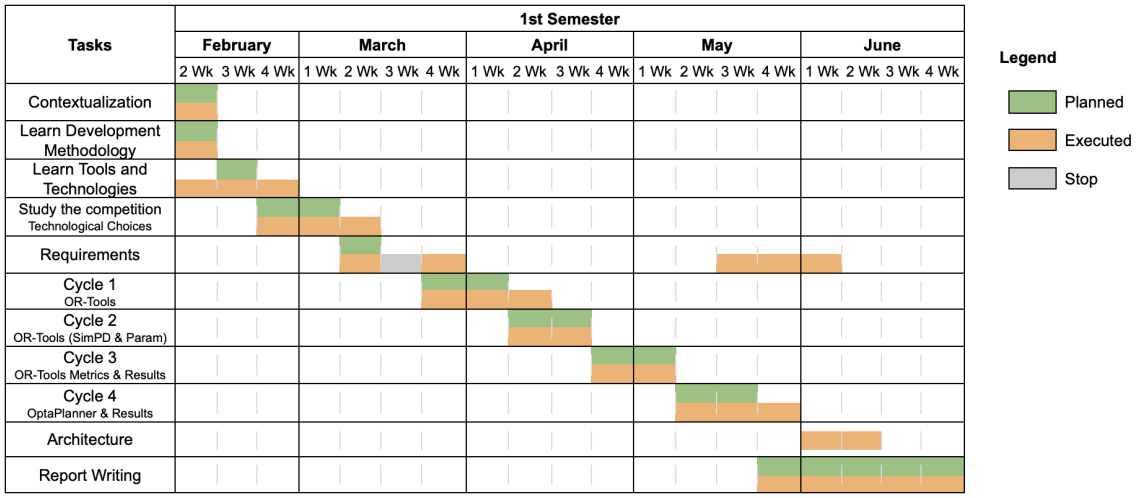


Figure 3.2: Planning for the first semester

The first week was dedicated to integrate in the project and learn about the team's working method. While **learning tools and technologies**, one week was not enough to understand how work plans can be generated, contextualize about the routing problem in optimization and the technologies used to solve it, so it took another week. The **study of the competition** involved identifying and comparing solvers to generate work plans. It took one more week than expected as the information about the tools was not sufficient to make a decision. A week was estimated for the **identification of requirements** to start the development, but took a week and a half due the thread of COVID-19. During the first week after the alert, no work was done and the company started to work remotely. What was planned was allocated for the following week.

At this stage, start the development process with a functional system would not be productive, more time to explore the solvers was invested. The planned sprints of 2 weeks were renamed as cycles to implement the passengers problem as prove of concept in *OR-Tools* and *OptaPlanner*, without delivering functional software at the end of the cycle. In **cycle 1**, a model for the passengers problem was implemented in *OR-Tools*. In **cycle 2**, variant with simultaneous pickup, simultaneous deliveries and parameterization from the data input was tested. In **Cycle 3**, results were compared with the *Sentilant* solution. Only **Cycle 4** was dedicated to *OptaPlanner*, because during the study of competing technologies *OptaPlanner* revealed that it was unable to find admissible solutions for a sequential pickup and delivery problem with 20 tasks even if running for hours for a problem that is solved by *JSprit* in less than a minute. With the learning of custom moves and help from the *OptaPlanner*'s community, custom moves were configured, thus obtaining solutions within an acceptable time.

The update of the requirements would be done during the sprints, but cycles were performed instead. So, three more weeks was dedicated for the requirements. As a last task of the first semester, a preliminary architecture was derived to present in the intermediate defense and a visualization of the system intended to build.

3.2.2 Second Semester

The plan defined in the first semester was followed for the second semester, and 2 additional sprints were made as part of this work for the integration with the FSM service used by the fuel model client. The work done and the planning is depicted in Figure 3.3.

The second semester started by reviewing the architecture and comparing the three solvers for the passengers problem. Based on the results, *JSprit* proved to be better in performance and in the quality of the solutions, so the *Sentilant*'s versions of the *JSprit* was chosen and time was invested to analyze the source code. The **first sprint** consisted in generate an initial problem instance and implement tanks states where the products were distributed by the compartments during optimization using an adaptation of the minimum sum sub-array algorithm. In the **second sprint**, several constraints were implemented, split-delivery and the periodic rests. The periodic rests required to change several parts of the *JSprit* core. During the **third sprint**, a PDF report generator was built to report the progress to the client and a presential meetings was held in the client's quarters. Based on the feedback, the sprint was readjusted, a better problem instance was generated and shifts were implemented, and two constraints were refined based on the feedback. KPIs about the tanks states e.g., percentage of empty space, total quantity transported were also collected. In the **fourth sprint** time was dedicated to improve the architecture, adapt source code from other *Sentilant* projects to build the system and interact with technologies such as Docker to learn how to containerize the Planning System. Another meeting was held

with the client, and new requirements emerged: avoid Ponte 25 de Abril, avoid travel large distances between simultaneous deliveries and a better distribution of the fuel by the compartments was requested. The **fifth sprint** was not enough to implement the avoid travel large distances and more days of the following sprint were dedicated, in the end it originated two constraints. In the **sixth sprint** the *API Gateway* was built using the latest versions of the required technologies with a development and a staging environment. Also, the documentation of the API was written for the tenant.

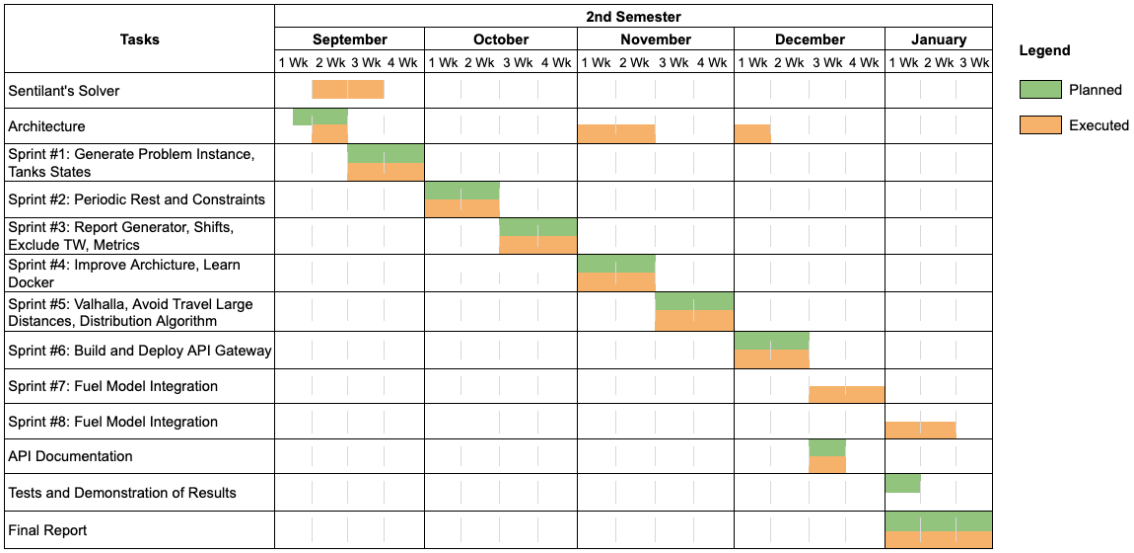


Figure 3.3: Planning for the second semester

The development of the Planning System ended in the sixth sprint. After than, my work continued in the company and to complete a development cycle for a tenant, time was dedicated to integrated the fuel model in a **seventh and eighth sprint**. The system used by the fuel model's client is tailored for him and does not use automatic work planning. Consequentially, the integration was developed for this client, until then no client of the *Sentilant*'s service used a model with tanks, shifts, periodic rests or capable to explicitly trace routes that avoid locations. Several integration problems were faced, for instance, the solver has the pickup convection, but for the client in its database, a cargo is a set of sequential pickups, this is, a cargo as one or more related deliveries, so non-existing cargo tasks are created after the optimization, or even destroyed in re-optimization scenarios, the system was not prepared for these operations. Further development is needed after these sprints, but at this stage the planning can be demonstrated.

3.3 Risks Management

The risk analysis is the process of identifying, prioritizing, and managing risk that threaten the accomplishment of a project objectives with success. The success is defined by the threshold of success, this is a set of conditions that must be met for the project to be considered successful. In the beginning of the second semester, the objectives of the project were revised, which originated one **threshold of success** for this project: *"Have a field-service planning system delivered as an API until the end of December 2020 that generates work plans for the fuel transport problem."* The followed risk management approach was informal, but with the threshold of success in mind.

Table 3.1 synthesizes the identified risks with decreasing ordered priorities and the miti-

gation strategy. The factors to measure it were identified. Then, the risks were prioritized in *Likelihood* (L) and the *Impact* (I). Both vary between Low (L), Medium (M) and High (H). The status identify whether the risk has already been overcome.

Whether there is no certainty that a risk can be completely avoid, mitigation strategies were defined. The followed risk management approach was informal where strategies to mitigate the risks were discussed with the team, keeping the threshold of success in mind.

Table 3.1: Risks Identification and Mitigation Plan

Risk	**Factor**	**L**	**I**	**Mitigation Strategy**	**Status**
Client refuses the planning because a specific requirement cannot be implemented.	Accept to develop a model without knowing if a requirement can be met The solver being used cannot satisfy the requirement.	M	H	1. Review production-ready solvers and elaborate a detailed comparison. 2. Test a proof of concept in the solvers: CVRPPDTW, custom soft and hard constraints; 3. Negotiate with the client alternative requirement(s).	Avoided
Client unsatisfied with the obtained work plans.	The client complaints about the results while using the solution in production.	H	M	1. Generate PDF reports illustrating the progress to present to the client and obtain feedback, during development; 2. Expose configurations in the API to allow adjust the model without redeploy the Planning System. e.g., change the loading time of a cistern; 3. Define and make available to the client a set of KPIs that allow him to know, before production, how the optimization will improve its cost/benefit rate.	
On the deadline date, the fuel model is developed, but there is no functional API.	Fine tunning the optimization takes longer than expected and time to develop the API is shorter	M	L	1. Start the design of the API as soon as possible; 2. Get help from the *Sentilant* team for bootstarting the development environment; 3. Start developing before the client approved the results from the optimization solver.	Avoided

In the work developed for this internship, objectives had to be defined, however the fuel model client as of January 2020 is not using the solution. Therefore, the second risk can only be considered avoided when the client is using the final solution and satisfied with the obtained work plans, however its mitigation was started in the beginning of the fuel model development with progress reports sent to the client to obtain feedback, and improve accordingly, thus avoiding dissatisfaction.

Chapter 4

Requirements

For the planning system, requirements are expressed as Architecturally Significant Requirements (ASRs), formally defined as the set of requirements that have significant influence over the architecture. Functional requirements in 4.1 from a high-level view using user-stories. Section 4.2 depicts ASRs as QAs scenarios, then prioritized and synthesized in the utility tree in 4.3. Section 4.4 depicts business and technical constraints.

4.1 High-Level Functional Requirements

Table 4.1: User-Stories to describe the high-level functional requirements

Id	Title	Description
1	Obtain a plan	As a **client**, I want to obtain a work plan for my specific problem, so that I can make logistic decisions for FSM.
2	Key Performance Indicators	As a **client**, I want to obtain metrics about the work plan, so that I can use it to manage our resources, fulfilling business goals.
3	Balance parameters	As a **client**, I want to be able to adjust some parameters, so that I could intervene in the optimization without strictly depending on the developers' decisions.
4	KPIs as developer	As a **developer**, I want to obtain KPIs, so that I could compare the quality of an obtained plan with other plans.
5	Update and add new tailored models	As a **developer**, I want to add new models and update the existing ones, so that I could serve different clients within the same system.
6	Monitor System Load	As a **system administrator**, I want to monitor the usage of the system, so that I can better manage and prevent loading pikes.

As an agile development process is followed, requirements were defined and planned verbally, through meetings with advisor Bruno Cabral, then documented as user-stories. The user stories are a non-formal, short and simple way to describe the requirements of the project from a stakeholder perspective. Thus, requirements can be handled without creating extensive formal requirements documents. A user-story has the following format:

"As a *<role>*, I want *<some goal>* so that *<some reason>*."

The roles are client, system administrator and developer. A client of the system is interested to obtain optimized work plans for FSM. The project manager represents the clients and also has a system administrator and developer. Before identifying QAs scenarios, user stories were gathered and presented in the Table 4.1. The *Id* is not indicative of

priority, but rather for quick identification.

4.2 Quality Attributes

During the capture of functional requirements through user-stories we responded to *who*, *what* and *why*, but we did not constraint *how* the system should work. Therefore, QAs were captured through scenarios using the methodology proposed in Bass et al. (2003) to describe QAs. According to the author, there is always a trade-off between different QAs. In this project, for instance, if we want the solver to find a better quality solution, there is a loss in terms of performance.

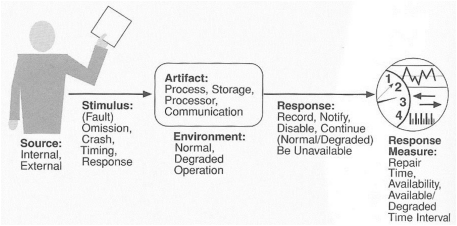


Figure 4.1: Parts of a QA scenario in *Software Architecture in Practice* from Bass et al. (2003)

A QA scenario is composed by six parts, the following is a brief explanation of each of the parts. In this way, the QAs of this system can be measured. For a better visualization, Figure 4.1 depicts the interconnection of each of the parts: i) **source of stimulus** is the entity that initiates the scenario by generate the stimulus, ii) **stimulus** is the event that initializes the scenario, iii) some **artifact** is stimulated. The component that receives the stimulus and produces a response. It may be the entire system, iv) **system response** is activity undertaken after the arrival of the stimulus, v) **response measure:** when the response occurs, it should be quantifiable in some fashion so that the requirement can be tested, and vi) **environment condition** the stimulus occurs within certain conditions. Puts the parts into context by describing the state of the system.

Following this approach, 10 QAs were captured presented in Tables 4.2 to 4.11 with a preceding a description:

"We develop tailored solutions for new clients by integrating new models in the system and updating the existing models according to the existing clients' needs. Developers add and update models during development, without modifying the remaining models code."

Table 4.2: Description of the QA Scenario 1: Extensibility

Extensibility	ID: **1**
Raw Attribute Definition	*Measure of the ability to extend a system and the level of effort required to implement the extension*
Source of Stimulus	Developer
Stimulus	Wants to update or add a new model
Environment Condition(s)	During development
Artifact	Source code
System Response	Model added or changed
Response Measure	Without modifying the remaining models code
User Story	Obtain a Plan (1), Update and Add New Tailored Models (5)

"Everytime a client gives feedback of preliminary results, the system should generate KPIs that allows him to assess the modification impact in the model, during development."

Table 4.3: Description of the QA Scenario 2: Solution Quality

Solution Quality	ID: **2**
Raw Attribute Definition	*Quality of the best solution found retrieved to the client*
Source of Stimulus	Client
Stimulus	Provides feedback about preliminary results
Environment Condition(s)	During development
Artifact	System
System Response	Model is modified accordingly
Response Measure	Updated results are obtained
User Story	Obtain a Plan (1), Key Performance Indicators (2), Balance Parameters (3)

"Some clients want to intervene in the optimization. The system allows the client to balance parameters (e.g., a weight factor) that impact the optimization, the unchanged parameters are considered with their default values."

Table 4.4: Description of the QA Scenario 3: Solution Quality

Solution Quality	ID: **3**
Raw Attribute Definition	*Quality of the best solution found retrieved to the client*
Source of Stimulus	Client
Stimulus	Submits a problem with additional changed parameters
Environment Condition(s)	Runtime
Artifact	System
System Response	Optimized plan is retrieved
Response Measure	The obtained plan reflects the changes compared to an optimization with the default values and provides feedback to the client that enables him to evaluate the effects of the parameterization
User Story	Key Performance Indicators (2), Balance Parameters (3)

"We expect our system to keep compatibility with the existing clients. The client will not be aware that he is accessing a different planning system."

Table 4.5: Description of the QA Scenario 4: Compatibility

Compatibility	ID: **4**
Raw Attribute Definition	*Ability of two or more systems (or components) to perform their required functions while sharing the same environment*
Source of Stimulus	Clients from the current FSM system of Sentilant
Stimulus	Performs Request
Environment Condition(s)	Runtime
Artifact	System
System Response	The system will answer as expected, and the client is not going to be aware that he is accessing a different optimization system
Response Measure	Client is not required to adapt his system
User Story	Obtain a Plan (1), KPIs (2), Balance Parameters (3)

"The client sends an identification of a problem not understandable as input to the API. The system spots the error in a field or the value out of range and retrieves a message."

Table 4.6: Description of the QA Scenario 5: Usability

Extensibility	ID: **5**
Raw Attribute Definition	*How easy it is for the user to accomplish a desired task and the kind of user support the system provides*
Source of Stimulus	Client
Stimulus	Sends a problem with a wrong parameter or value
Environment Condition(s)	Runtime
Artifact	System
System Response	A informative error message is retrieved
Response Measure	After the system spots the first error, a message is built and retrieved to the client.
User Story	Obtain a Plan (1), KPIs (2), Balance Parameters (3)

"When modifying the source code, we want to ensure that we did not introduce any new errors. After updating or introducing new models in the system, a battery of tests is run to verify that new errors have not been introduced."

Table 4.7: Description of the QA Scenario 6: Testability

Testability	ID: **6**
Raw Attribute Definition	*Ease with which software can be made to demonstrate its faults through testing*
Source of Stimulus	Developer
Stimulus	Perform unit tests
Environment Condition(s)	At development time
Artifact	System
System Response	Each model is successfully verified if it's working correctly
Response Measure	For each model is verified that errors were not introduced.
User Story	Obtain a Plan (1), Balance Parameters (3), Update and Add New Tailored Models (5)

"We expect our system to take a similar performance to the existing planning system in Sentilant. The runtime to optimize a problem does not take 15% longer than the current solution, given inputs up to 50% above the typical size of the client's problem, considering all the constraints to satisfy the same problem."

Table 4.8: Description of the QA Scenario 7: Performance

Performance	ID: **7**
Raw Attribute Definition	*It's about time and the software system's ability to meet timing requirements.*
Source of Stimulus	Client
Stimulus	Wants to obtain an optimized work plan
Environment Condition(s)	Runtime
Artifact	System
System Response	Optimized work plan retrieved
Response Measure	The runtime to optimize a problem does not take 15% longer than the current *Sentilant*'s solution, for problems with dimensions up to 50% above the typical size of the client's problem.
User Story	Obtain a Plan (1), Balance Parameters (3), Update and Add New Tailored Models (5)

"A client submits a problem much larger compared to its typical size, then receives an error

message in the generation of the plan. The system administrator sets a limit for each model and receives a notification whether the limit is exceeded."

Table 4.9: Description of the QA Scenario 8: Scalability

Scalability	ID: **8**
Raw Attribute Definition	*Scalability is the ability of the system to handle load increases without decreasing performance, or the possibility to rapidly increase the load.*
Source of Stimulus	Client
Stimulus	Submits a problem much larger than usual
Environment Condition(s)	Runtime
Artifact	System
System Response	Retrieves an error message
Response Measure	Whether the problem submitted to solve exceeds the limit set by the system administrator for the correspondent model an error message is generated, then sends a notification to the system administrator
User Story	Obtain a Plan (1), Monitor System Load (6)

"For answering a new client, a model is added to the system. Therefore, the increase in the number of simultaneous calls to the system should not introduce latency increases of more than 15%."

Table 4.10: Description of the QA Scenario 9: Scalability

Scalability	ID: **9**
Raw Attribute Definition	*Scalability is the ability of the system to handle load increases without decreasing performance, or the possibility to rapidly increase the load.*
Source of Stimulus	Developer
Stimulus	New model added
Environment Condition(s)	Runtime
Artifact	System
System Response	Respond to an additional new client
Response Measure	The increase in the number of simultaneous calls to the system should not introduce latency increases of more than 15%
User Story	Obtain a Plan (1)

"The client needs to provide his credentials to access the API. The system denies users with wrong credentials or inactive accounts."

Table 4.11: Description of the QA Scenario 10: Security

Security	ID: **10**
Raw Attribute Definition	*The measure of the system's ability to withstand unauthorized attempts to use data or services, providing access to legitimate users.*
Source of Stimulus	Client with a credential
Stimulus	Accesses the system
Environment Condition(s)	Runtime
Artifact	System
System Response	Grants access to the API
Response Measure	Whether the account is active and the credential is valid, otherwise denies access to the API.
User Story	Obtain a Plan (1)

4.3 Utility Tree

Table 4.12: Utility Tree representing the ASRs captured for this project, the attributes are sorted by business value in descending order, where the most valuable is at the top.

P	Quality Attribute	Attribute Refinement	ASR
1	Extensibility	Source code structure	Add and update models during development, without modifying the remaining models code. **(H,H)**
2	Solution Quality	Client feedback	A client gives feedback of preliminary results, the system generates KPIs to allow the client to assess the modification impact in the model. **(H,L)**
3	Solution Quality	Optimization intervention	The work plan must reflect the influence of the parameters set by the client in relation to the default values. **(H,L)**
4	Compatibility	Interface	Existing clients access our planning system without being aware of it. **(M,M)**
5	Performance	Optimization duration	The optimization duration on this system should not be more than 15% slower than the current *Sentilant* solution. Considering the typical size of the client's problem. **(M,H)**
6	Security	Authentication	Only clients with authenticated and active accounts are allowed to submit and receive optimized work plans. **(M,L)**
7	Usability	Consistent messages	Upon detection of an invalid problem identification, indicate to the client the location and the reason for the error. Messages should be consistent across all clients. **(M,L)**
8	Scalability	Limit the input size	System administrator can set a limit to the maximum size of the client's input to prevent abuse of the system. **(L,L)**
9	Scalability	Latency	The increase in the number of simultaneous calls to the system should not introduce latency increases more than 15%, as new models are added and new clients can access the system. **(L,M)**
10	Testability	Verification	Ensure errors were not introduced in the models after modifying the source code. **(L,M)**

Utility tree in Figure 4.12 synthesizes the QAs sorted by priority in descending order, where the most important is on the top. This prioritization was conceived through a vote between me and the advisor. Each of us had 75 points to distribute between each scenario, the more total points given to a scenario, the higher the priority. A large amount of points was used to avoid draws.

The ASRs present a notation with two letters, in which (L=Low, M=Medium, H=High). The first letter represents the **business value** that is automatically mapped from the priority. We define the three most voted as must-have requirement with High and the remaining are Medium, except the last three that are ranked as Low. The second letter represents the **architectural impact**: i) High: meeting the ASR profoundly affect the architecture; ii) Medium: meeting the ASR somewhat affect the architecture; iii) Low: meeting the ASR have little effect on the architecture.

The **extensibility and the solution quality are the most important QAs** for this

system. Extensible due to the multi-tenant characteristic of the system, since a model can be expensive to develop, however, it is intended that the addition of new models does not affect the results of the existing ones. Solution Quality because the system should produce optimized work plans tailored for each client.

Facilitate adding new models (1) and optimization duration (5) are the ASRs with the highest architectural impact in our perspective. The mechanisms provided by the solver to implement a model may change drastically, requiring the developer to understand too many technologies. Moreover, there is a preference to use a single solver to implement all the problems and reduce the time-effort over time, than fragment models in many different applications due to algorithm's performance, for instance.

Without reverse engineering a solver and taking it to its limits to meet performance requirements, it may imply to trade the extensibility and the solution quality for it. Therefore, optimization duration was ranked as an ASR with a high architectural impact.

Presenting KPIs to receive client feedback (2) during development is important. However, the way those KPIs are presented only make sense when aggregate with the correspondent identification of the problem and the plan in a human-readable way. A PDF report was proposed and used to present preliminary results to the client.

The *Sentilant*'s FSM system submits problems and receives work plans for its clients through a queue service, but there are clients that interacts through an *API*, so Compatibility (4) pretends external systems access our new system with minimal change. However, in case of the *Sentilant*'s FSM system a new integration has to be developed, thus ranked as medium architectural impact.

Security (6) and usability (7) have a medium business value, but they are straightforward to ensure without impact other QAs. Scalability and testability affect the architecture as new models are integrated, however they reveal a lower business value comparing to the remaining QAs.

For clarification, in performance the typical size is the size of a problem instance from a real client. In the absence of real problem instances, we have to be guided by the size indicated by the client.

4.4 Business and Technical Constraints

The architecture is commonly constrained by a set of business and technical constraints. Those constraints are perceived as assumptions, limiting the design-decisions. In beginning of the second semester, 1 business constraints and 1 technical constraints was pinned, respectively:

- **Scheduling:** The Planning System API to solve the fuel problem must be finished by December 2020;
- **Free software licencing:** All the software required to develop the components must be free of charge, presenting a license e.g., Apache License, allowing to legally use it in a production environment without expose the source code.

Moreover, there is a preference to use Java and Python programming languages, as these are the main languages used in the backend stack of *Sentilant*, being more maintainable in this regard. However, there is no imposed restriction on using other languages.

Chapter 5

Architecture

An overview of the system is presented in 5.1 with the C4 model. The communication mechanisms to serve plans in 5.2, a description of the required persistent data in 5.3, the main components involved in a optimization inside the solver in 5.4, the structure of a problem instance and the work plan in 5.5, and the deployment environments in 5.6. Throughout the chapter, architectural decisions are justified with the QAs from utility tree in section 4.12.

5.1 System Overview

The high-level architecture of the system is designed following the Simon Brown (2011)'s C4 Model offering 4 levels of detail to visualize an architecture (C1: Context, C2: Containers, C3: Components, C4: Classes). Designed only with the first two levels, furthermore the author says they are enough for the majority of the projects.

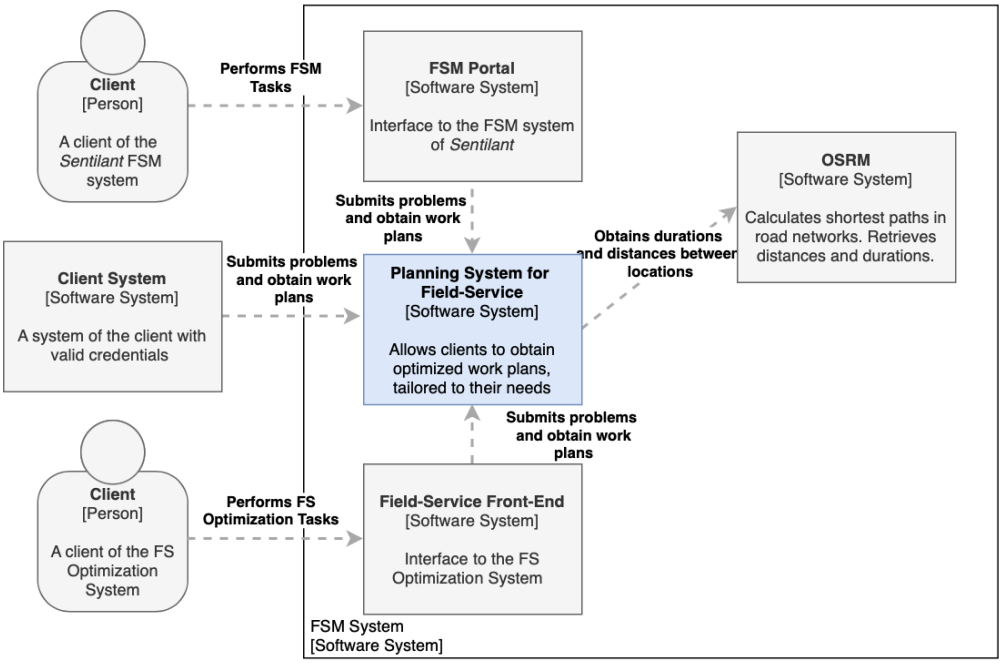


Figure 5.1: System Context Diagram: clients can directly interact with the planning system or the system is invoked by a component of the *Sentilant* system.

The **context diagram** defines the scope of the system, focusing on people and abstracting the software system from technologies. Figure 5.1 depicts the context with three use-cases for the system. In the first use-case, the external clients can integrate the access to the planning system in their applications, accessing directly to it. In the second and third usage context the system is integrated as part of the *Sentilant* system, the client doesn't directly access it, but other applications from the FSM system does. The Open Source Routing Machine (OSRM) is a service running in *Sentilant* to calculate routes and estimate distances and durations between locations given their coordinates.

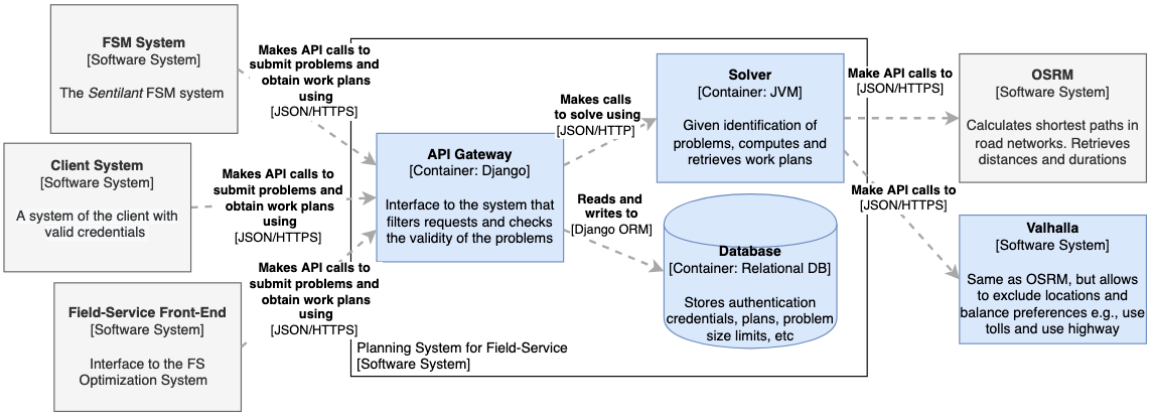


Figure 5.2: System Container Diagram for the Planning System

With **container diagram**, the high-level technologies to be used are defined and how the containers communicate with each other inside the system. A container is any separately runnable/deployable unit. In Figure 5.2 the three main containers of the system are visualized: *API Gateway*, database and solver.

The core of the development was centered on the **Solver**, the component was deployed as a server application responsible to process an optimization request at a time, and generate optimized work plans. The chosen solver is *JSprit*'s project from *Sentilant*, since the other solvers showed to be more than 60% slower than *JSprit* and finding worse solutions, **{Performance (5)}**, for more information see results in section 7.1.

The **API Gateway** is the interface to access the system, and filters the problem instances **{Usability(7)}**, whether an instance is invalid returns a <message><error> message e.g., *"Task scheduled datetime end before start"*, *"Task 3 expects scheduled_ start_ datetime to start before scheduled_ end_ datetime!"*. As the optimization can take time, this service is offered asynchronously — a client can submit a problem and obtain the plan later. The *API Gateway* is a REST service, for dealing with asynchronous requests the *Celery* is used with the in-memory database *Redis* and message broker *RabbitMQ*. Sysadmin decisions as change the maximum problem limit or add new credentials take effect in this container.

The **Database** persist data for the *API Gateway* explained in section 5.3. Moreover, to deal with asynchronous requests after retrieving the work plan from the solver, it's saved on the *Database*.

For the fuel model, it was necessary to avoid routes that passes through *"Ponte 25 de Abril"*, as the **OSRM** did not allow it, an alternative called **Valhalla** can and was configured and deployed in this work, it can be accessed from a public domain.

The possibility of being able to scale horizontally, that is, more instances of the solver running in parallel or even adding other solvers, is not excluded. For this internship, two solver instances run in parallel.

5.2 Communication Mechanisms

In the current solution of *Sentilant*, the solvers are accessed through a *RabbitMQ* message broker. With this Planning System, we wrapped the solver and provide the *API Gateway* as the only interface. Three types of requests are provided: synchronous, asynchronous and webhooks. The synchronous type is attractive, but both the receiver and the sender of the request must wait with an active HTTP connection until receiving the response. In the asynchronous type, a planning request is made and the HTTP connection is terminated, but the customer never knows when the work plan is ready. The third type is similar to the second, but *API Gateway* submits the work plan via HTTP to an address specified by the client. Figure 5.2 depicts the communication mechanism. The *API Gateway* and point-to-point messaging queues provide the bridge between the external systems and the solver's instances. It used a *Celery* background worker to receive the plans from the queue.

The *API Gateway* keeps the same schema as the existing *JSprit* input and output **{Compatibility (4)}**, although it requires to modify the current architecture of the external systems e.g., FSM System to integrate the access to the API.

A *RabbitMQ* message broker is used to transmit messages between the *API Gateway* and the solver. Thus, hundreds of solver instances can run in parallel, however the system should just be ready to scale and comply with the **{Scalability(9)}** in a med-long-term.

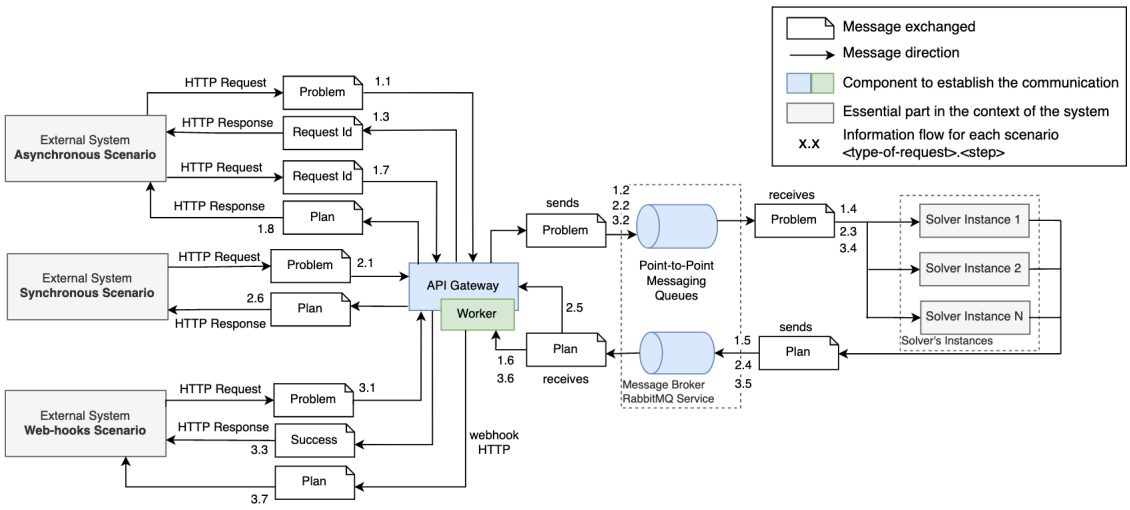


Figure 5.3: Communication mechanism describing the transported messages.

5.3 Persistent Data

The system only persists data for technical reasons. Keeping information is not required for the business, the strict intention of this system is to produce work plans given problem instances. A *Postgres* database is used to persist the latest requests to support asynchronous operations, control in terms of security, scalability and reproducibility of errors. *Postgres* is chosen because it's the most known database by the *Sentilant*'s team. The relational model is depicted in Figure 5.4. Token-based authentication is used **{Security (6)}** to protect the system against unauthorized requests, the token is stored in the database as token field in *Key*. Clients and it's respective tokens are added in the database manually. The *ProblemLimit* defines limits for the dimension and run-time of the problems **{Scalability (8)}**, thus preventing computational waste and abuse of the system. There is a global limit and optionally a limit set for the token that overrides a global limit.

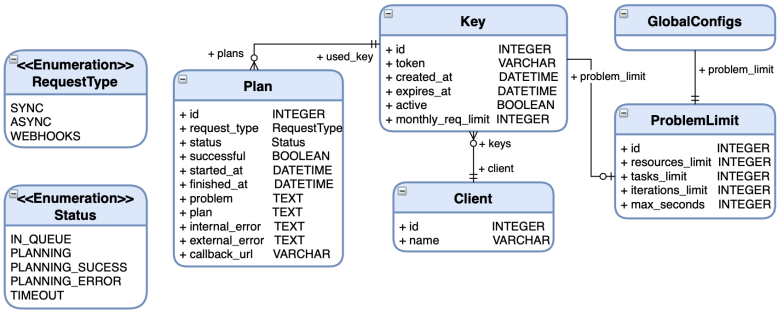


Figure 5.4: Relational model of the database accessed by the *API Gateway*.

5.4 Solver Pipeline

The solver project from *Sentilant* that uses *JSprit* is the solver chosen for this architecture. To consolidate the pipeline of the Solver from a high-level perspective, first a sequence diagrams describing the main objects interactions and the messages exchanged is presented, then a second for multiple-optimizations.

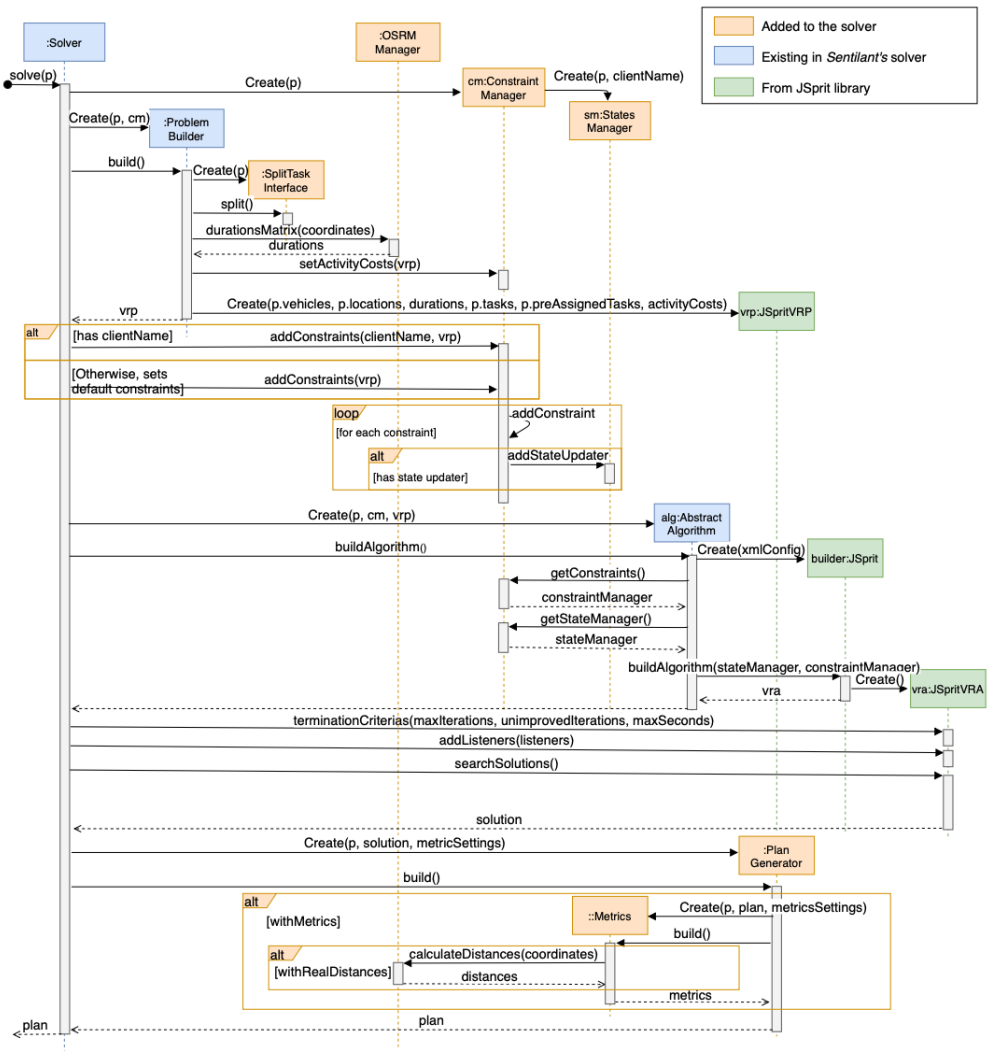


Figure 5.5: High-level operations from setting up the *JSprit* solver until generating the work plan for an optimization. The classes marked in orange were added.

Sequence diagram in Figure 5.5 describes a view of the *Sentilant*'s project that consumes the *JSprit* solver. The **Solver** is the entity that recurs to the *JSprit* library to solve a problem and generate a work plan. The **Problem Builder** an entity that build the problem data in *JSprit*. The **Split Task Interface** provides a way to divide tasks with large quantities into smaller tasks with a fraction identification; **OSRM Manager** provides durations and distances between locations. For **Constraint Manager**, there are many custom configurations that change the behavior of the model, this is not limited to hard and soft constraints. Some of the constraints depend on custom states and dynamic costs. The **State Manager** keeps the state related implementations such as the cisterns states. **JSpritVRP** represents the problem for the *JSprit*. **JSprit** represents the algorithm of the *JSprit*, termination criterias, and listeners that triggers during the optimization process; The **Plan Generator** generates the work plan giving a solution from the *JSprit*. For the **Metrics** it can have different settings, for instance, distances are slow to obtain from the OSRM and Valhalla, so it can calculate the distances based on the euclidean distances or based on the velocity formula.

A discussion was held at *Sentilant* on whether metrics should be calculated on *API Gateway*, on an independent service or on the solver. Since *JSprit* runs in single-thread, and all the information is contained in the solver, it was decided to be calculated in the solver. Moreover, one of the objective of the *API Gateway* is to be stateless, the component should requires as little maintenance as possible, therefore this is another argument for this logic to belong to the solver. **{Solution Quality (2)}**

The fuel model requires more than one optimization due to drivers working in shifts, because the start of the second-shift depends on the end of the first-shift. Therefore, an executor was added to externally handle more than one optimization and calculates the metrics in the end after appending the plan. Sequence diagram described in Figure 5.6.

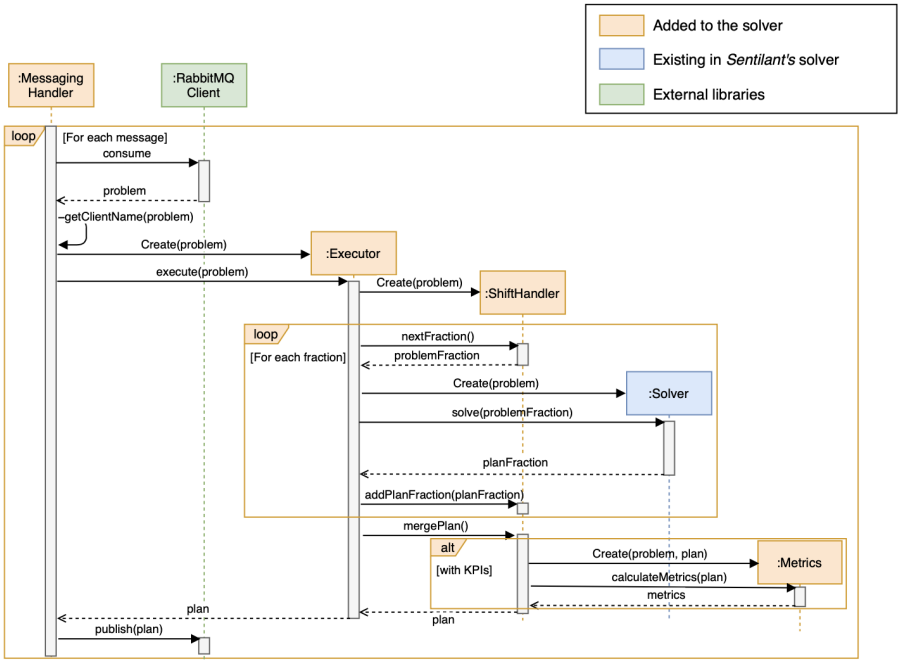


Figure 5.6: Behavior of a solver instance from reading a problem from the Message Broker until publishing the response. Problems with shifts require more than one optimization.

5.5 Messages exchanged

The two messages exchanged by the API are the problem and the resulting work plan. As the company's solver was used, the required attributes for the fuel problem were added, and all fields that have no effect on this model have been removed.

The problem input is described in Figure 5.7, the representation of the products were added as **Product** and **ProductRequest**, the representation of the tank of each resource as **Compartment** of a **Tank** which contains the loading time of the cistern. The **PeriodicRest** represents the rest time every driving time. Some of the functionalities were kept as the **TaskSettings** that is a structure that accepts arrays of tasks, where an array represents tasks that must be performed sequentially, **Skill** where a task with the skill only can be performed with resources containing the same skill, also the algorithms settings were kept. Regarding fields, the **costManager** indicates how transitions between locations are calculated (e.g., euclidean distance), the **speedFactor** allows to decrease or increase the transitions time and distances from the cost manager, the **useHighwaysFactor** and **useTollsFactor** allow to indicate Valhalla the preference for highways and tolls, the Valhalla is already prepared to receive this parameters. **More details about the parameters are provided in the next chapter 6, while presenting the optimization model.**

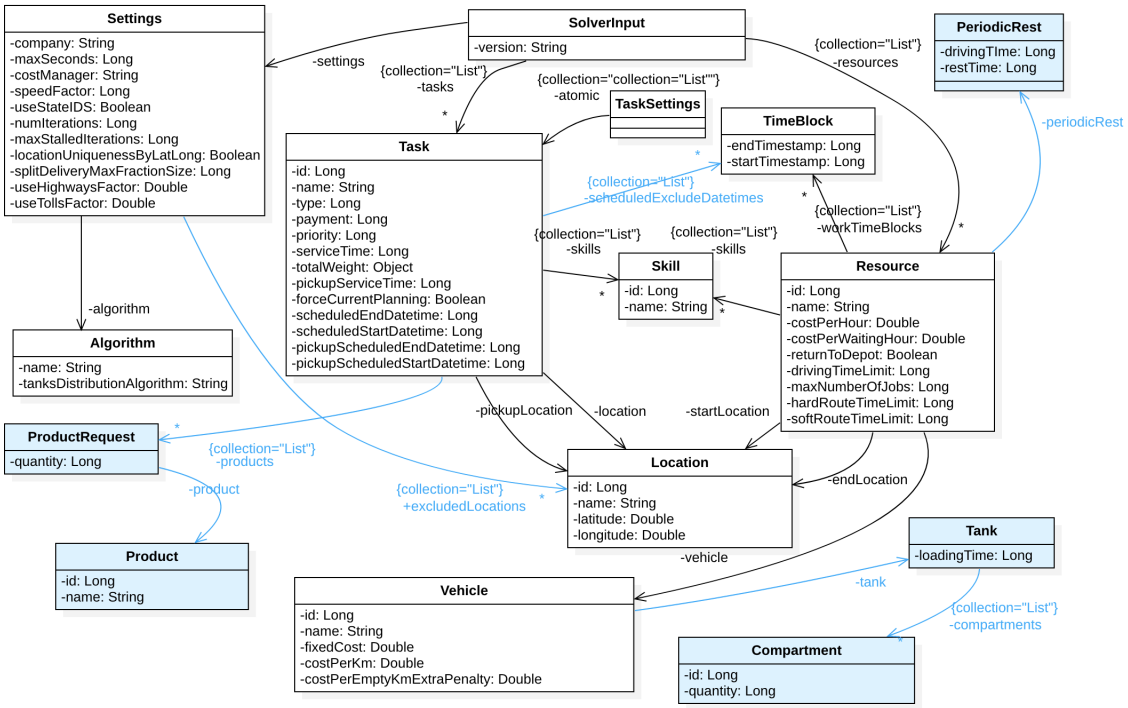


Figure 5.7: Problem input representation with the added classes in blue.

The work plan structure for the fuel model is represented in Figure 5.8. In addition to a fraction (i.e., part of a task), and the shift identification, **4 types of metrics** were extract and appended: **RouteMetrics** are about the route, **PlanMetrics** obtained from the route metrics constituting the KPIs, **AlgorithmMetrics** related with the optimization algorithm, and **UnassignedReasons** that tells how many times each hard-constraint was broken. Metrics allows to external clients measure the modifications impact on the work plan **{Solution Quality (3)}**.

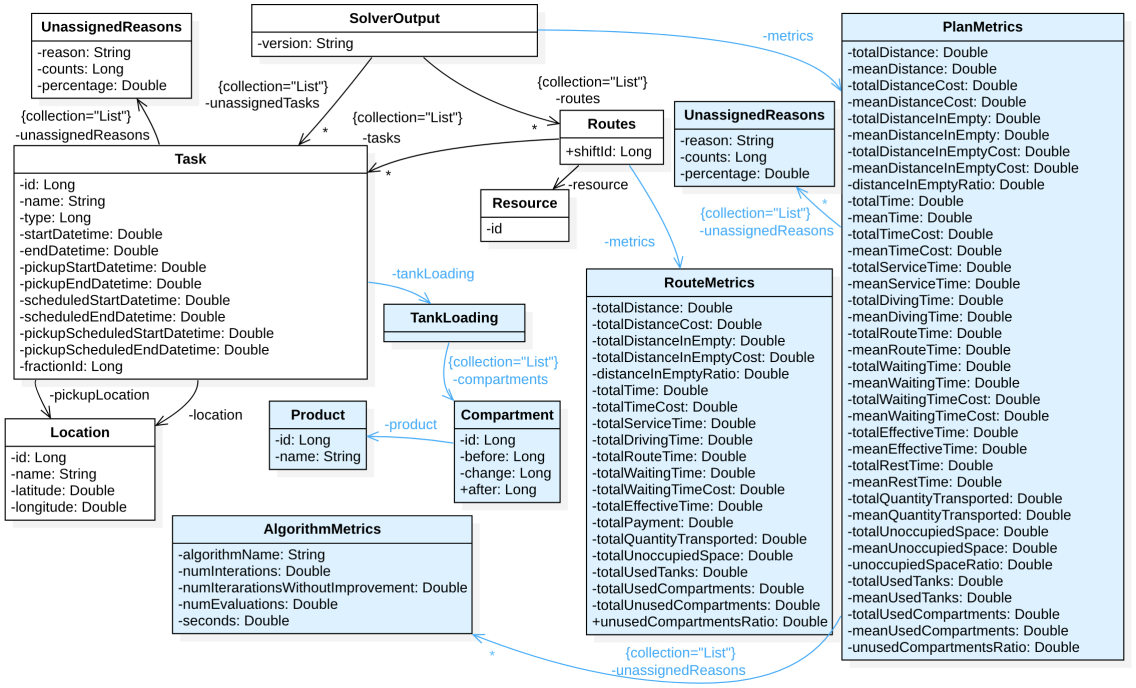


Figure 5.8: The work plan output represented with the added classes in blue, also the *fractionId* and *shiftId* fields representing the assigned part of the task if divided and the shift identification.

5.6 Deployment Environments

The company uses docker (*docker.com*) to manage the deployment environments. *Sentilant* uses at least three deployment environment in its workflow: development, staging and production. Two deployment environment were used, development and staging. In the **development environment** the containers are: a **RabbitMQv3.8.9**, a **PostgresSQL v13** relational database, an in-memory database **Redis v6.0.9**, a **Django v3.12.2** web application for the *API Gateway*, a **Celery v4.1.1** worker to read responses from the queue, a **Readbeat Server** (*github.com/sibson/ redbeat*) that stores scheduled *Celery* jobs. For the **staging environment**, the **NginX v1.19** is used, two instances of the solver are ran in two containers with **JDK v13**, and debug mode and any profile software is turned off. All the services are containerized with docker on a server machine running Ubuntu v18.04.4 operating system with a i7 3770 CPU up to 3.9 Ghz and 4.5Gb of RAM. **Valhalla** was launched on a cloud server with 8 GB of RAM and 4 threads. By default, Valhalla accepts up to 50 locations to calculate the asymmetric matrix, the configuration was changed to 250, moreover Valhalla is also containerized (*github.com/gis-ops/docker-valhalla*) using Portugal and Spain maps (*download.geofabrik.de*).

The architecture built in this work does not comply with **{Extensibility (1)}**. In the development of the system, the behavior of the solver was modified, including in the **JSprit** core, which may have change the results of the existing models of *Sentilant* e.g., passengers model, so we were unable to guarantee this QAs despite having been voted as the most important. As for testability, some unit tests were implemented to ensure that whenever the solver is compiled, new errors were not introduced, mainly code sections that involve computations during optimization e.g., fuel distribution. Moreover, the deployment environments help to ensure **{Testability (10)}**.

Chapter 6

Optimization Models

As explained in the previous chapter, the planning system only provides the fuel transport model, but two optimization models were developed in this work. In 6.1 is described the **passengers model** developed in *OR-Tools* and *OptaPlanner*, then in 6.2 is described the **fuel transport model** developed with the *Sentilant*'s solver. For each model is described a problem definition to transmit the requirements, a formalization of the problem, and the implementations that involved developing custom functionalities (excluding configurations) in the solver(s).

6.1 Passengers Transport Model

In 6.1.1 is described the passengers model requirements and the formulation of the optimization model in section 6.1.2 based on Holborn et al. (2012), depicting the requirements of the passengers transport problem. The problem's model and the local search movements implemented in *OptaPlanner* to improve the performance of the model for pickup and delivery are presented in 6.1.3 and 6.1.4, respectively. This is not required for *OR-Tools* and *JSprit* since both have a native implementation for pickup and delivery that prevent breaking VRPPD hard constraints.

6.1.1 Problem Definition

The client receives requests to transport people from **accommodations** to the **airport** and vice-versa. A task consists of a request to **transport one or more people**, the pickup is denominated as departure and the delivery as arrival. Departures have **short-period time-windows** e.g., 10 minutes, whereas arrivals have no restrictions.

The client's main objective is to **allocate as much tasks as possible**, while having control over the optimization. The amount of tasks can vary from just a few tasks to more than 200 for a single day, and the number of resources employ by the client varies from just a few drivers up to 40. Each drivers has an associated vehicle with a variable number of seats most are 5-seat cars, but only **2-3 seats are available**, and can only perform a request at a time. There is a concern for passengers satisfaction, **they should fell comfortable when being transported**, the more free seats available in the vehicle the better.

A driver usually starts in his home and usually ends in the same location, and have an

associated **cost per hour**, **maximum driving time** and **maximum working time**. Despite drivers having a maximum work limit, the client sometimes wants to reduce the length of the assigned working time to each driver, while assigning work to all of his drivers. The associated vehicle has a **cost per kilometer** and a **fixed cost** for being used. Furthermore, the client's experience in his business reveals that **drivers do not appreciate to travel many kilometers with the vehicle empty** i.e., driving while not performing any task. Regarding **tasks, some have more priority than others**, and an associated **payback to be tuned by the client**.

Currently, there is a solution in the company for a client with a similar problem as told in the introduction. For this problem, the client himself adjusts the parameterization and sometimes re-optimize several times the same instance to improve the work plan. This is a special client, in a way that most of the *Sentilant*'s clients have less control over parameterization.

6.1.2 Problem Formulation

To define our version of CVRPPDTW for the passengers problem, let $V = \{v_0, v_1, .., v_n\}$ be a set of locations where n is odd, composed by three sub-sets $V = M \cup N^+ \cup N^-$. Let $M = \{v_0, v_1, .., v_{K*2-1}\}$ be the set defining the start and end location of each vehicle $k \in K$ in the route where a pair $\{v_{2k}, v_{2k+1}\}$ represents the start and end location and $|M| = 2K$. Let $N = \{v_{|M|}, .., v_n\}$ be the set of pickup and delivery locations where N^+ denotes the pickup locations and N^- the delivery locations. A pair of locations $\{v_i, v_{i+1}\} \in V$ where $i >= |M|$ and i is even, the location $v_i \in N^+$ denotes the pickup and $v_{i+1} \in N^-$ the respective delivery location. Therefore, $N^+ \cup N^- = N, N^+ \cap N^- = \emptyset, |N^+| = |N^-|$.

A location $v_i \in V$ has an associated time window $[tMin_i, tMax_i]$ which represents the earliest and the latest time that a service can start at location i. A service at location i takes s_i time to be performed. In case of $v_i \in M$ represents the work block of the respective vehicle k. A vehicle can arrive before $tMin_i$, so let A_i be the arrival time and W_i the wait time. If $A_i < tMin_i$, then the vehicle has to wait W_i at location i, where $W_i = tMin_i - A_i$. For $v_i \in N^+$, let $q_i > 0$ be the demand to be pickup and the $q_{i+1} = -q_i$ demand to be delivered. Each task has a payback and a priority, so for delivery locations $v_i \in N^-$, let $p_i >= 0$ be the payment amount and $pr_i \in \{1, 2\}$ be the priority of the task where 1 is *normal* and 2 is *high*. Let t_{ij} be the duration between location i and j, and d_{ij} be the distance between location i and j, in addition both duration and distance are asymmetric i.e., the value from i to j can differ from j to i.

Let Q^k be the maximum capacity of the vehicle k, $tHard^k$ be the maximum time that can be traveled (i.e., when the vehicle is moving, excluding the wait time W_i and service time s_i), $tDrvHard^k$ the maximum driving time, T^k be the cost per hour, C^k be the cost per kilometer, E^k be the cost in empty per kilometer, F^k be the fixed cost, the comfort bound $qUpper^k$ be the desired capacity range to keep along the road for the vehicle k, such that $qUpper^k$ respects $qUpper^k <= Q^k$, and J^k be the maximum number of assigned tasks to k. Also, let $tSoft^k$ be a soft maximum working time, for each vehicle k.

The **variables** are: i) **decision variable** $x_{ij}^k : V \times V \times K$ representing sets of ordered locations, each set represents a route of the vehicle k, ii) **cumulative variables** that update depending on the value of the decision variable x_{ij}^k.

$$x_{ij}^k = \begin{cases} 1, & \text{if vehicle } k \text{ goes from visit } i \text{ to visit } j \\ 0, & \text{otherwise} \end{cases}$$

$$notA_j = \begin{cases} 1, & \text{if } \sum_{k=0}^{K-1} \sum_{i \in N} x_{ij}^k = 0, \text{ pair with delivery } j \text{ is} \\ & \text{not assigned (over-constraining)} \\ 0, & \text{otherwise} \end{cases}$$

$$D_i = max\{A_i, tMin_i\} + s_i, \text{ departure time from visit } i$$

$$tWork_i = \text{ working time until visit } i$$

$$tDrv_i = \text{ driving time traveled until visit } i$$

$$l_i^k = \text{ load of the vehicle k at visit i, after service i}$$

$$a^k = \begin{cases} 1, & \text{if } \exists \sum_{i,j \in V} x_{ij}^k = 1, \text{ then vehicle } k \text{ is assigned} \\ 0, & \text{otherwise} \end{cases}$$

$$qAboveU_i^k = \begin{cases} 1, & \text{if } l_i^k > qUpper^k \text{ for vehicle } k, \text{ after service } i \\ 0, & \text{otherwise} \end{cases}$$

In addition to the problem data, let $\{w_0, w_1, w_2\}$ be a set of parameterizable weights to adjust the penalization of the objective function, where w_0 influences the maximum working time $tSoft^k$, w_1 the comfort bounds and w_2 the payback. The **objective function** is:

$$\begin{aligned} \min f(x) = & \sum_{k=0}^{K-1} a^k [F^k + (tDrv_{2k+1} - tSoft^k) w_0] \sum_{i,j \in V : i \neq j}^{|V|} x_{i,j}^k [t_{i,j} \cdot T^k + d_{ij} (!l_i^k \cdot E^k + C^k)] \\ & + \sum_{i=|M|}^{n} qAboveU_i^k (l_i^k - qUpper^k) w_1 + \sum_{i=|M| : i\%2 \neq 0}^{n} notA_i (p_i \cdot pr_i) w_2 \end{aligned}$$

The $!l_i$ means if the vehicle is empty the value 1 is yield, otherwise 0, thus firing the distance in empty penalization. Decomposing the objective function: i) fixed cost $a^k \cdot F^k$, ii) maximum working time cost $a^k \cdot (tDrv_{2k+1} - tSoft^k) \cdot w_0$, iii) duration on the route $t_{i,j} \cdot T_k$, iv) distance in empty traveled cost $!l_i^k \cdot d_{ij} \cdot E^k \cdot w_2$; v) distance traveled cost $d_{ij} \cdot C_3^k$, vi) comfort upper bound $qAboveU_i^k (l_i^k - qUpper^k) w_1$, and vii) tasks not assigned / payback / priority $notA_i (p_i \cdot pr_i) w_2$.

The **constraints** are as follows. The constraints 6.1 ensures that a task location is either visited once or not visited/assigned. Constraints 6.2 ensures the vehicle departs from a given start location and 6.3 ensures the vehicle arrives at the end location if the vehicle is assigned, otherwise 0. Constraint 6.4 ensures a vehicle is either assigned or not assigned. Constraints 6.5 ensures whether a vehicle arrives to a visit, then the vehicle must also depart from that visit. Constraints 6.6 ensures the pickup and delivery visits are assigned to the same vehicle k. Constraints 6.7 ensures the non-transgression of the maximum tasks that can be assigned. Constraints 6.8 and 6.9 ensure the capacity constraints are

met. Constraints 6.10, 6.11, 6.12 ensure time windows, precedence of sequential pickup and delivery visits and perform before end time of the work block and visits. Constraints 6.13 and 6.14 ensures the vehicle respects the maximum driving time and distance.

$$0 \leq \sum_{k=0}^{K-1} \sum_{j=0:i\neq j}^{n} x_{ij}^{k} \leq 1, \forall i \in V \tag{6.1}$$

$$0 \leq \sum_{j=|M|}^{n} a^{k} \cdot x_{k*2,j}^{k} \leq 1, k \in K \tag{6.2}$$

$$0 \leq \sum_{i=|M|}^{n} a^{k} \cdot x_{i,k*2+1}^{k} \leq 1, k \in K \tag{6.3}$$

$$\sum_{j=|M|}^{n} x_{k*2,j}^{k} - \sum_{i=|M|}^{n} x_{i,k*2+1}^{k} = 0, k \in K \tag{6.4}$$

$$\sum_{i=0:i\neq h}^{n} x_{ih}^{k} - \sum_{j=0:j\neq i}^{n} x_{hj}^{k} = 0, \forall h \in N, k \in K \tag{6.5}$$

$$\sum_{p=0}^{n} x_{pi}^{k} - \sum_{d=|M|}^{n} x_{d,i+1}^{k} = 0, \forall i \in N^{+}, k \in K \tag{6.6}$$

$$\sum_{i\in N^{+}} \sum_{j\in N^{-}} x_{i,j}^{k} <= J^{k}, \forall k \in K \tag{6.7}$$

$$l_{i}^{k} \leq Q^{k}, \forall i \in V, k \in K \tag{6.8}$$

$$x_{ij}^{k} = 1 \Rightarrow l_{i}^{k} + q_{j} = l_{j}^{k}, \forall i, j \in V, k \in K \tag{6.9}$$

$$\begin{gathered} x_{ij}^{k} = 1 \Rightarrow D_{i} + t_{i,j} = A_{j} \Rightarrow A_{j} \leq D_{j} \Rightarrow D_{i} \leq D_{j}, \\ \forall i, j \in V, k \in K \end{gathered} \tag{6.10}$$

$$A_{i} \leq A_{i+1}, \forall i \in N^{+} \tag{6.11}$$

$$x_{ij}^{k} = 1 \Rightarrow A_{j} \leq tMax_{j}, \forall i, j \in V, k \in K \tag{6.12}$$

$$tWork_{k*2+1} \leq tHard^{k}, k \in K \tag{6.13}$$

$$tDrv_{k*2+1} \leq tDrvHard^{k}, k \in K \tag{6.14}$$

This formulation as at least one feasible solution which in the worst case no task is assigned. The **solution representation** is given by a set of routes $R = \{r_0, r_1, .., r_K\}$ where r_k is represents by a subset of V ordered locations. If $a^k = 1$, then r_k is not empty and the vehicle is assigned. Otherwise, r_k is empty and k is not assigned.

6.1.3 Domain Model

In the *OptaPlanner* framework, the representation of a problem has to be created known as domain model. The implemented domain model is explain without mathematical notation.

The solver provides a use case model to solve the VRPTW, and can be extended to solve the passengers transport problem. A *Pair* class was added to represent a pickup and a delivery, and the missing data added to the *Visit* and *Vehicle* classes. For instance, the driving time limit and the cost per hour were added to the model. Figure 6.1 illustrates the domain model, where *VehicleRoutingSolution* represents the problem and contains the data and variables involved in the optimization process. The instances of *Vehicle* and *Visit* are the objects that can be modified by the solver declared as *@PlanningEntity*, and the remaining are declared as *@ProblemFact* that cannot change during optimization. The routes are represented by doubly linked lists, where the *@PlanningVariable* is changed by the local search algorithm, and the shadow variables are updated accordingly, facilitating access to data. Not shown in the figure, but as part of the model, custom state updaters were implemented to update the driving, working duration, the load of the vehicle, the arrival date time and the index of a visit in a route after every local change.

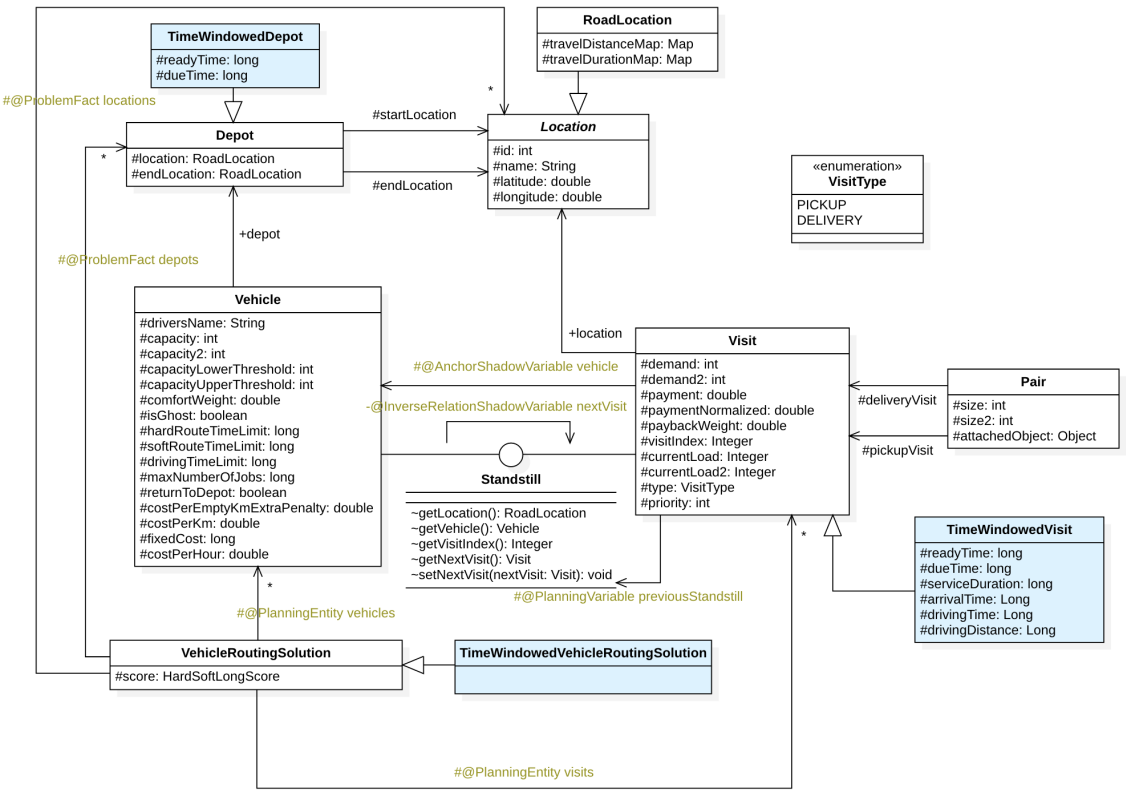


Figure 6.1: Domain Model implemented in *OptaPlanner* adapted from a VRPTW example, the extended classes in blue color for the time-windows logic

6.1.4 Optimization Algorithm

A local search algorithm explores the search space by applying local changes moving to neighbor solutions. Six operators are applied to explore the neighborhood in *OptaPlanner*: change, swap, change pair, swap pairs, sub-route change, and sub-route swap. Examples of the movements are described in Figures 6.2 and 6.3. The examples consider two distinct routes, although moves also occur within the same route i.e., intra-route. Any of these operators performs inter and intra-route movements, except inter-route movements $r_a\, r_b$: $a \neq K \wedge b \neq K$ in the bag that keeps the unassigned visits.

A **change operator** moves a visit v_i from one place to a different place. A **swap operator** exchanges two visits v_i and v_j. In the Figure 6.2b is depicted a basis for the change operator movement and in Figure 6.2c for the swap operator movement.

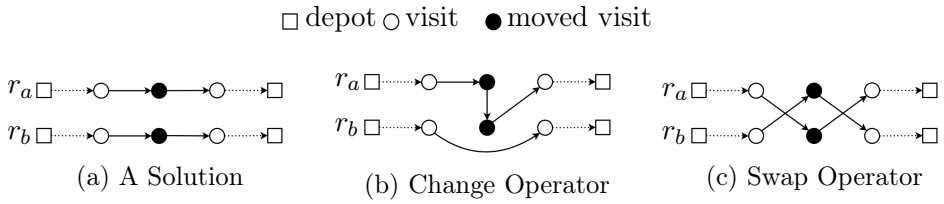


Figure 6.2: Simple local search movements used in the passengers model: (b) and (c).

Movements in Figure 6.3 take advantage of visits forming pickup and delivery pairs. The **change pair operator** selects a random pair $\{v_i, v_{i+1}\}, i \in N^+$ then chooses a random visit $v_j, j \in V$ and create a set with potential previous visits P to assigned to the delivery v_{i+1}. The v_i and any visits after v_j are inserted into P. To execute the movement, the previous visit of v_i is assigned such that $v_j \leftarrow v_i$, and the previous visit of v_{i+1} a random visit $v_r \in P$ such that $v_r \leftarrow v_{i+1}$. The **swap pairs operator** selects two random pairs $\{v_i, v_{i+1}\}, i \in N^+$ and $\{v_j, v_{j+1}\}, j \in N^+$ where $i \neq j$. Then, pickups v_i and v_j are swapped and the same for the deliveries v_{i+1} and v_{j+1}. In the Figure 6.3b is depicted a basis for the change pair movement and in Figure 6.3c for the swap pairs movement.

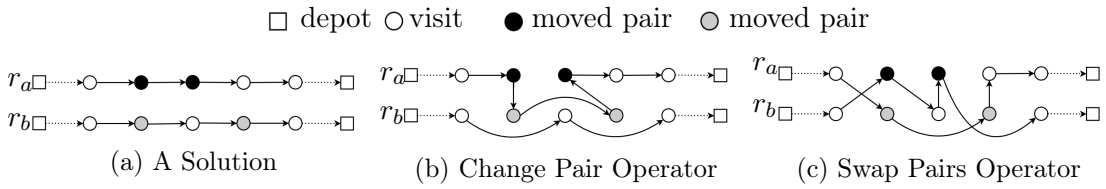


Figure 6.3: Local search movements taking advantage of the pickup being assigned before a delivery and belonging to the same route. Example of move (b) and (c)

6.2 Fuel Transport Model

Problem definition provided in 6.2.1 describing the fuel model requirements, a formulation of the optimization problem in 6.2.2 excluding constraints, and a mapping from the problem definition to implementation in 6.2.3 considering the architecture of *JSprit* and client's PE. Then, an explanation of the effort to implement the requirements in 7.2.3.

6.2.1 Problem Definition

The supply of Gas Stations (GSs) is made by carriers that transport fuel from **Cargo Centers (CCs)** to **GSs**. One of the largest Portuguese national carriers, which supplies GSs of a fuel brand, does **manual planning** to allocate tasks to its employees. A task consists of an order with specific **quantities of fuel** that must be delivered within a time-window established by the GS. Some GSs have **several time-windows**, or we can think in just one time-window where certain intervals are excluded. The information regarding these time-windows is known for each GSs and usually does not change. However, the majority of GSs accepts deliveries at any time of the day.

The carrier's main objective is to **reduce the number of cisterns** used to satisfy all orders. On a typical day, the carrier has **26 tractors** available, in which each tractor has a cistern attached. The majority of the cisterns have **6 tanks**, being able to transport up to 36 thousand liters of fuel. The capacity is higher, but for safety reasons the tanks are limited to a lower capacity (36 thousand liters). There are also cisterns with 1 or 3 tanks, but they are used less often in this problem. Of these tractors, 23 of them are driven by drivers who **work from 8 AM until 6 PM**. Each of the remaining **3 tractors is driven by 2 drivers who work in shifts**. The first shift starts at midnight until 3 PM and the second shift from 1PM until 3 AM the next day. A driver can **work up to 10 hours**, and sometimes some of the orders cannot be fulfilled, so the schedule can be extended to 12 hours, however this work period must be avoided. By this, we mean that these 10 working hours limit can be relaxed. **Every 4 hours of driving** (with a legal maximum of 4 and a half) the driver must take a **45-minute rest**, and it does not count as working time. A driver is also limited to a **maximum of 9 to 10 hours of driving**.

A **driver starts in a CC** and has to end the route in the same CC. Loading a cistern, regardless of whether one or all tanks in the cistern are loaded, the **loading time is 1 hour**. For delivery, it can take **up to 1 hour to unload the fuel from the cistern**, depending on the number of tanks to be emptied. Typically, if all the tanks or almost all tanks will be emptied in a GS, it takes 1 hour. Otherwise, it takes half an hour to 45 minutes. **In the first loading, the cistern must be loaded in the CC where it starts**. After the first loading, a cistern can be loaded at another CC. Furthermore, a cistern can only be loaded in a CC, and to be loaded it must be empty.

Some orders require the transportation of a quantity of fuel **exceeding 36 thousand liters**. These orders have to be delivered by more than one cistern. There is no obligation that the planning system has to allocate all **fractioned tasks**, it can allocate just a fraction i.e., term used for a part of an order when it's divided into smaller parts. Orders that do not exceed capacity must be delivered all at once, as GSs do not appreciate receiving more than one cistern on the same day at different times of the day.

The CCs belong to companies such as Galp and Repsol. The CCs are already assigned prior to planning to the respective orders, however it is possible to load at another CC of the same company (e.g., Galp). If a change occurs the CC's owner (e.g., Galp) has to be notified of the change. The change of CC can be made at most to 3 or 4 orders. This change is only accepted if it results in a better plan, however it must be avoided.

In addition to the objective of reducing the number of cisterns, the carrier wants an automatic planning to **assist the employee who plans manually**, as the manual planning is something that is carried out by carrier employees and takes several hours to build a plan for problems like this, that involves allocating all the tasks/orders to the drivers.

6.2.2 Problem Formulation

This problem is a CVRPPDTW with multiple-compartments, in the literature known as Multi-Compartment VRP (MCVRP) with fixed compartment sizes. To define our version of the *MCVRP*, Passengers Model's formulation is inherited and the modifications explained.

The comfort bounds $[qLower^k, qUpper^k]$ and E^k are not considered. Multi-compartments are considered, so for $v_i \in N^+$, let $q_i^p > 0$ be the demand of the product type p to be pickup and the $q_{i+1}^p = -q_i^p$ demand to be delivered. Let Q^{kc} be the maximum capacity of each compartment c of the vehicle k. Let TC be the number of compartments of each vehicle k and TP the number of fuel (i.e., product) types. A location $v_i \in N$ has an associated time window $[tMin_i^w, tMax_i^w]$, where w represents the time-windows of the pickup or delivery i. The start and end locations $v_i \in M$ only have one time window w corresponding to the working block of the driver. Let P be a periodic rest, and D be a time interval, where every D driven time by a vehicle k there is a mandatory rest during P time.

The **variables** introduced in the passengers problem are kept, except $qAboveU_i^k$, $dDrv_i$ and l_i^k. The additional **cumulative variables** that update depending on the value of the decision variables x_{ij}^k are: $tWork_i$ time elapsed from start until arrive at visit i, and l_i^{kc} load of each compartment c of vehicle k at visit i, after service i.

The *JSprit*'s algorithm has two types of soft constraints evaluated in different steps: cost function and insertion cost. The following **objective function** is the cost function. Consider M as a sufficient large number that outweighs any possible total sub-cost from the first term of the cost function.

$$\min cost = \sum_{k=0}^{K-1} \sum_{i,j \in V : i \neq j} x_{ij}^k [t_{ij} \cdot H^k + d_{ij} \cdot C^k] + \sum_{i \in N^-} notA_i \cdot M$$

For the MCVRP constraints useful to understand our implementation, they are presented in the cistern states hard constraint in the next section.

6.2.3 Mapping to Implementation

The mapping is divided into 4 parts describing exclusively what was added in this work to the *Sentilant*'s solver, accompanied by Preferences Elicitation (PE) to provide facts about our decisions. In **pre-processing** is described the functionalities added that act outside of the optimization model, but still part of the fuel model. Then, the added **hard constraints** and **soft constraints**. Also, there are variable that are stateful during optimization, so **states** described the implementations that modifies and/or adds states to the model.

Pre-processing

Fraction tasks: Given $T = \{q_0^p, .., q_n^p\}$ representing the demands of the GSs n, where p represents descending ordered products. Let F be the max fraction size. Every GS n that requests a total quantity $\sum_{i=0}^{n} > F$, the order is divided into at least two tasks, where the $t-1$ new tasks have F total quantity. The task is divided as many times as, and the fuels are divided in descending order.

PE: Fractioning orders should be avoided, since the GSs doesn't appreciate receiving more than one delivery on the same day. However, tasks above the capacity of a cistern have to be divided and supplied in two or more separated deliveries.

Shifts: Each driver has an associated time-window $[tMin_i, Max_i]$ representing the work block of the driver k. There are vehicles shared between drivers that work in shifts. Thus, let j be a second shift driver and $D_j = max\{A_i, tMin_j\}$ be the start time of the second shift, where A_i is the end of the respective first shift.

Two optimizations are proposed to solve this problem. There are 3 drivers working with shifts from midnight up to 3 PM and is known that are several requests that can only be fulfilled during the night, the following configuration shows the best results. Let K_1 be the set of first shift drivers, K_2 be the set of second shift drivers, K_3 be the set of drivers without shifts, and T be the tasks.

- **First optimization:** The drivers K_1 of the first shift and the tasks $T_1 \subseteq T$, where deliveries $tMax_i \leq tMax_k$ are selected for the first optimization. As result of the optimization, let $U_1 \subseteq T_1$ be a set of unassigned tasks and R_1 be the routes;

- **Second optimization:** The drivers K_2, K_3, and the tasks $T_2 \cup U_1$ not selected for the first optimization plus U_1 are selected for the second optimization. As result of the second optimization, let R_2 be the routes and U_2 the unassigned tasks;

- **Merge Results:** The resulting routes for a given problem are R, where $R = \{R_1, R_2\}$ and the unassigned tasks $U = U_2$.

Hard Constraints

Cistern states: Given a candidate job $\{p_i, d_j\}$ to be inserted on a route, and l_i^p the previous state of the cistern, were p is the compartment index, calculate the updated state of cistern l_i^p after inserting the demand q_i^p. Let $r_a = \{p_0, .., p_{n-1}\}$ be the sub-route of sequential pickups p_i and $L_a = \{l_0^p, .., l_{m-1}^p\}$ be the compartments states. Update the states of the compartments with the added products, if cannot insert all the products the constraint is not fulfilled. This constraint also requires a state updater and a state manager to keep track of the cisterns loading.

Consider a set of n products $\{q_0, ., q_{n-1}\}$, a set of m sorted capacities in descending order $\{Q_0, .., Q_{m-1}\}$, and c the compartment index. Let the **variable** $l^{cp} \in [0; q^p]$ be the quantity of a product or a part of it added in compartment c and the decision variable $y^{cp} = 1$ if product p in compartment c, otherwise 0. The **objective** $minf(y) = \sum_{c=0}^{m-1} \sum_{p=0}^{n-1} y^{cp}(Q_c - l^{cp})$ pretends to a find solution that minimizes the amount of unoccupied space in the cistern. The constraints are:

$$0 \leq \sum_{p=0}^{n-1} y^{cp} \leq 1, \forall c \in \{0, .., m-1\} \tag{6.15}$$

$$\sum_{p=0}^{n-1} y^{cp} \cdot l^{cp} \leq Q_c, \forall c \in \{0, .., m-1\} \tag{6.16}$$

$$\sum_{c=0}^{m-1} y^{cp} \cdot l^{cp} = q_p, \forall p \in \{0, .., n-1\} \tag{6.17}$$

$$\sum_{p=0}^{n-1} y^{cp} \cdot l^{cp} \geq \sum_{p=0}^{n-1} y^{c+1,p} \cdot l^{c+1,p}, \forall c \in \{0, .., m-2\} \tag{6.18}$$

Constraint 6.15 ensures there is only one product or part of it is in a compartment. Constraint 6.16 ensures the capacity is not exceeded for each compartment c. Constraint 6.17 ensures each product is allocated in the compartments. Constraint 6.18 ensures the largest compartments carry the largest quantities. Moreover, if $\sum_{p=0}^{n-1} q_p > \sum_{c=0}^{m-1} Q_c$ holds, the problem instance has no feasible solutions.

PE: The unloading of the compartments is trivial. But, for the loading process there are several approaches. The client is interested in minimizing the total number of cisterns in use, and if there is any empty compartments after loading the cistern, they must be the smallest ones and the larger quantities of fuel have to be inserted in the larger tanks. This is, the fuel arrangement is important for the tractor's stability.

The rationale behind this decision is related to the stability of the tractor and the cistern. Each of the compartments has a different dimension. If only one compartments is loaded, it must be the one with the highest capacity, because it will generate less instability in the tractor. If there are 2 occupied compartments, the two with greater capacity must be used, and if the quantities differ, the larger must have the greater amount of fuel. If there are n occupied compartments, the same rule applies.

Prevent loading while not empty: This constraint was already implemented;

PE: The same geographical location e.g., Aveiras can have more than one CC, for instance a Galp CC and a Repsol CC. A driver can only load in only one CC at a time.

First loading on start location: Given a candidate job $\{p_i, d_j\}$ to be inserted on a route, the geographical location of the start of the route v_k i.e., a CC, and the first delivery of the route after the insertion d_0. Let v_i be the geographical location of the inserted pickup, if the pickup p is inserted before d_0 and $v_k \neq v_i$ the constraint is not fulfilled, otherwise is fulfilled.

PE: As a general rule of the client, the first loading is done at the start location. To relax the solutions generated by the solver, it is considered that after the first loading, the driver can load the cistern in another CC as long as it ends at the CC where the route started.

Working time limit: Given a candidate job $\{p, d\}$ to be inserted on a route r_a, $tHard^k$, and $tWork_k$ of the vehicle k after the insertion. The condition $tWork_{2k+1} \leq tHard^k$ has to hold, otherwise the constraint is not fulfilled.

PE: The periodic rest does not count as working hours, however waiting and service hours do count. This is an important detail since the calculation of the working time $tWork_k$ differs from client to client.

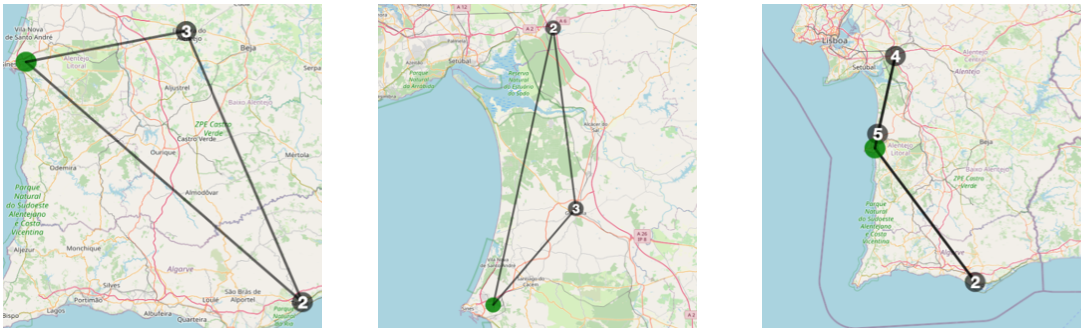
Soft Constraints

Loading time: Given a sub-route set of sequential pickups of any route $r_a = \{p_0, .., p_n\}$ and the loading time constant L. The first pickup p_0 has a service time $s_i = s_i + L$, while $r_a \setminus \{p_0\}$ have 0 has the service time.

PE: In reality, loading a tank or a complete cistern does not take exactly the same time. It is debatable, but usually all tanks start to be loaded at the same time and the client considers 1 hour for all loading times in its plannings, and told us to also consider it.

Deliver closest GSs to the CC first: Given a candidate job $\{p_{new}, d_{new}\}$ to be inserted in a route, and r_a the sub-route of deliveries where d_{new} is candidate to be inserted. If $|r_a| \leq 1$, the penalty does not apply, otherwise obtains the distance between the CC and the assigned delivery d, and the inserted delivery d_{new}. Whether the insertion of d_{new} is after p and $dist_{new} \times x < dist$, gives a penalty equivalent to the distance between d and d_{new}. Otherwise, if the insertion is before p, and $dist_{new} > dist \times x$ gives a penalty equivalent to the distance between d and d_{new}. The factor $x = 1.10$ relaxes the application of the penalty, when $dist$ and $dist_{new}$ are close there is no need for penalty.

PE: The client does not appreciate that tractors deliver to the furthest GSs first. Given a loaded cistern to supply more than one GS, the nearest GSs must be supplied first, preventing scenarios as in Figure 6.4. Those are the shortest path routes. When delivering to the nearest GS first slightly more kilometers are traveled, which makes the solver choose these solutions instead. In other cases hard-constraints are broken, so turning impossible to invert the route.



(a) (b) (c)

Figure 6.4: Routes (a), (b) and (c) delivers first the furthest GS then when the tractor is turning back the cistern still have fuel to deliver in another GSs. Routes (a): CC (214.5Km) → 2 (160.6Km) → 3 (69.2Km) → CC; (b): CC (93.3Km) → 2 (62.4Km) → 3 (44.7Km) → CC; (c): CC → 2 → CC (93.3Km) → 4 (93.4Km) → 5 (16.9Km) → CC. Distances obtained with *Valhalla* engine.

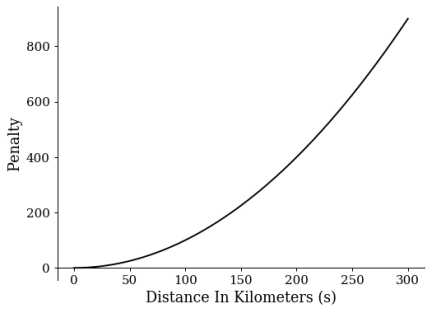


Figure 6.5: Penalty function $\frac{1}{100}s^2$

Avoid travelling large distances between simultaneous deliveries: Given a candidate job $\{p, d\}$ to be inserted on a route. Let $r_a = \{d_0, .., d_n\}$ be the ordered sub-route of deliveries where the delivery is being inserted, and $dist_{i,j}$ the distance between any two location i and j. If $|r_a| \leq 1$ the penalty does not apply, otherwise let $s = \sum_{i=0}^{n-1} dist_{i,i+1}$ be the sum of the distances between deliveries in r_a, and $\frac{1}{100}s^2$ be the insertion penalty.

Thus, it penalizes the insertion of jobs during the recreate step, proportional to the distances traveled between deliveries in the sub-route of ordered deliveries where the delivery

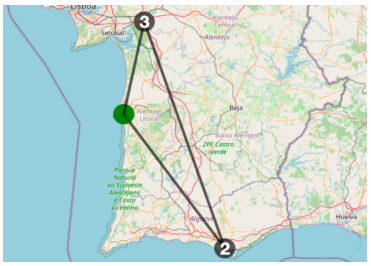


Figure 6.6: Route starting in Sines CC, goes to (2) to supply a GS, travels 213.9 km to supply a GS in (3), and returns to the CC.

is being inserted. The penalty function (polynomial $\frac{1}{100}s^2$) was adjusted several times, exponential functions were also tried, but generated very large penalties and heavily impaired the generation of clockwise routes.

PE: The client does not appreciate that tractors travel a long distances between sequential deliveries. Even if the plan generates a solution with a greater total distance traveled. There is a high preference to avoid situations as depicted in Figure 6.6. The tractor travels back a very long distance to supply the second GS.

States

To add custom states in *JSprit* state manager and state updaters are used. The state manager keeps stateful information e.g., variables during optimization that are not part of the solution representation. A state updater triggers if route and/or activity .

Cistern states: Representation of the cistern state managed by the state manager, and a state updater to update the load of the cistern.

Periodic Rest: Every time a transition $t_{i,j}$ occurs from i to j and a multiple or multiples of D are reached with the $tDrv_j^k$ in relation to $tDrv_i^k$, count multiples reached as X, $t_{i,j} = t_{i,j} + P \cdot X$.

6.2.4 Discussion

This dissertation does not contain a chapter dedicated to the implementation, so this section explains to the reader the student's effort allocated to implement each of the requirements for the fuel model, thus in table 6.1 is summarized the effort. The **first column** orders each of the requirements temporally, in the order in which they were implemented. The **second column** indicates approximately how many days were spent implementing the requirement. The **third column** explains if the requirement needed to be adjusted to the problem, so tunning required to allocate time and trial error. The **fourth column** indicates if unit tests were written to test the functionality, and the **last column** provides a subjective view of the difficulty to implement the requirement in *JSprit*. The difficulty is classified as *Low*, *Medium* and *High*, the first means the implementation was straightforward requiring at most 2 days, *Medium* required to analyze the existing code thoughtfully and to make some changes after the initial implementation. The *High* is similar to medium, but required to change the *JSprit* core and included more development.

For requirement (1) to implement in *JSprit* required to write a state updater, a constraint, the representation of the tanks, a tank state manager required by the solver and the algorithm to add products in the cistern, an adaptation of the minimum sub-array sum

Table 6.1: Summary of the effort allocated to the implementation of the requirements.

Requirements	Days	Tunning	Unit Test	Difficulty
1. Cistern States	7-8		✓	High
2. Periodic Rest	5-6			High
3. Working Time Limit	2			Medium
4. First Loading on Start Location	1-2			Low
5. Shifts	3	✓		Medium
6. Fraction Tasks	< 1	✓	✓	Low
7. Deliver Closest First	3-4	✓		Medium
8. Avoid Large Distances Sim. Del.	3-4	✓		Low
9. Improved Cistern States	3-4		✓	Medium

was implemented to find the set of free compartments that yield the minimum unoccupied space after introducing a product. The representation of the tank had to be added to *JSprit* to be able access it, since a new object representation (i.e., tanks) was introduced in the problem definition of the *JSprit*. The main difficulty was implementing all these components and made them work together.

For requirement (2) required to change several parts of the *JSprit* core that could not be extended, from changing how $t_{i,j}$ is calculated, take periodic rest into account in time-windows state updaters and reshape some constraints. A non-existing driving time variable had to be passed around in several methods to conceive this functionality as native. This functionality requires to change several unit tests that were ignored, however based on black-box tests the results were always right. To update the existing tests, it would take several days of work, in which several tests would be necessary to be re-written.

For requirement (3) the existing constraint used for the passengers model was re-written. The periodic rest had to be included in the calculations and required to lead with the pickup and delivery insertions to calculate both the current and the driving time after the insertion, using the following method. The *nextAct* is the job (i.e., pickup or delivery in this problem) that is trying to be inserted. Moreover, the *prevActDrivingTime* is one of the new parameters add to the *JSprit* core because of the periodic rest, and it's not part of the default version of *JSprit*. In the implementation of (3) it was avoided to use cycles, although tempting.

ConstraintsStatus fulfilled(JobInsertionContext iFacts, TourActivity prevAct, TourActivity newAct, TourActivity nextAct, double prevActDepTime, double prevActDrivingTime)

The requirement (4) was the simplest of the list, requirement (5) required tunning because of the shift schedules, some experiences were done trying to obtain the best results for this problem. Requirement (6) was implemented with a recursive algorithm, in the first implementation if a task was divided approximately the same quantity was distributed by the new tasks, although it was changed to the version in the previous section. Requirements (7) and (8) required to implement a way to calculate the smallest angles and to obtain sets of simultaneous deliveries or pickups giving the route and a delivery or pickup, respectively. This concept of "simultaneous" was used to improve the cistern state in requirement (9), instead of update based on the previous state, all the products were re-introduced to yield better solutions and a recursive distribution algorithm was written, in some cases this constraint was broken, but in reality the products would fit in the cistern.

As a **reflection**, if these requirements had been implemented in *OptaPlanner* less effort would have been required, however, it would be necessary to invest in performance. In addition, no implementations were found in these tools (e.g., Github) with an answer to a problem similar to this.

Chapter 7

Results

The first two sections present the performance and quality of the solutions obtained by the solvers. The comparison between *OR-Tools*, *OptaPlanner* and the *Sentilant*'s solver for the passengers problem in 7.1, in which the *Sentilant*'s solver yielded better results, thus used to develop the model for the fuel transport problem with results in 7.2. Section 7.3 illustrates the request time of an optimization and summaries the API, the feedback report presented to the client is explained in 7.4, and the results of the integration with the FSM service used by the fuel model client in 7.5.

7.1 Passengers Transport Model

First a statistical description of the problem instances used to compare the models is given in 7.1.1, the experimental setup explained in 7.1.2, a comparison between KPIs in 7.1.3, and a discussion about the trade-offs of choosing a solver over another for this problem in 7.1.4 as take home message to the reader.

7.1.1 Data Description

A set of 30 problem instances were randomly obtained from the optimization records of the client between 2018 and 2020, for privacy reasons no information about the locations is provided. The mean number of **tasks** is 124.90 (SD=122.15) and **resources** is 28.13 (SD=16.76). Let start by the resources, in total there are 844 resources in the dataset. Time expressed in <hours>:<minutes>, the **work time blocks** has a mean of 19:27 (SD=05:33; min=03:34; median=20:09; max=25:59), **driving time limit** 08:45 (SD=01:17), **working time limit** 11:03 (SD=02:50), **fixed cost** 0 (SD=0.00), **maximum number of tasks** to be assigned to the resource 223 (SD=100.44), **cost per hour** 2.05 (SD=3.11), **per km** 1.09 (SD=0.39), **per empty km** 1.00 (SD=0.00). For the capacities, the client uses two types, **occupation** to prevent sequential pickups 1.00 (SD=0.00), and **passengers** 5.10 (SD=2.11) as the limit of people that can be transported with **upper bound** of 559.95 (SD=5345.50; min=3.00; median=4.00; 75%=8.00; max=52018.00).

For the tasks, in total there are 3747 tasks in the dataset. Time expressed in <hours>:<minutes>:<seconds>, the **pickup time-window** has a mean of 00:05:19 (SD=00:03:44), **delivery time-window** 11:59:49 (SD=00:11:29), **pickup service time** 00:08:29 (SD=00:04:11), **delivery service time** 00:05:31 (SD=00:01:09), **payment** 35.89 (SD=26.20), and **priority** 1.0 (SD=0.0). For the required capacity, mean **occupation** of 1.00 (SD=0.00),

and for **passengers** 3.06 (SD=1.84). The weights $\{w_0, w_1, w_2\}$ have means of $\{0.41(SD = 0.31), 0.45(SD = 0.27), 0.66(SD = 0.22)\}$.

7.1.2 Experimental Setup

Solvers were ran in a server machine running Ubuntu v18.04.4 operating system with a i7 3770 CPU up to 3.9 Ghz and 4.5Gb of RAM, configured to run up to 30 minutes. The configuration of the optimization algorithm in the solvers is depicted in Table 7.1. For the termination criteria, *OR-Tools* only returns feasible solutions, but *OptaPlanner* may return an infeasible solution, so the algorithm was configured to only stop whether a feasible solution is found and the solution is not improving. Different constructive heuristics and local search algorithms were tried, the GLS and LA showed better performance.

For the *OR-Tools*, the parameterization related to moves remained unchanged, whereas for *OptaPlanner* an implementation with custom moves described in section 6.1.4 was done to improve the performance of the model due to breaking pickup and delivery related constraints. Two additional moves where used that grabs a random sub-route i.e., set of sequential locations and swap or move them to other random place.

Table 7.1: Environment Configuration for the Passengers Problem

	Jsprit	**OR-Tools**	**OptaPlanner**
Threads	1	1	4
Reproducible	Yes	Yes	No
Termination Criteria	500 iterations ∨ 200 unimproved iterations	10 unimproved iterations	5 unimproved seconds ∧ 10 unimproved iterations ∧ No hard constraints broken
Constructive Heuristic	Best Insertion	Local Cheapest Insertion	First Fit Decreasing
Local Search	R&R	GLS	LA (AcceptanceSize=64)
Moves	—	—	2 single, 2 sub-route, 2 custom pair

7.1.3 Comparison Results

To quantify the difference against *JSprit*, the percentage change (equation: 7.1) was calculate for the relevant KPIs. Table 7.2 reports an overview of the results, where *JSprit* is 2.78 times faster than *OR-Tools* and 1.63 times faster than *OptaPlanner*, while finding solutions with more tasks assigned that consequentially yields more monetary payment.

$$\text{Percentage Change} = \frac{\Delta V}{|V1|} \times 100 \tag{7.1}$$

The total traveled distances and durations is lower in *OptaPlanner* than the other solvers, but when compared by the assigned ratio, it assigned fewer tasks. The amount of assigned tasks is the most important KPI, which diminishes the quality of the work plans obtained by *OptaPlanner* significantly. As a last observation, *OR-Tools* presents the lowest distance in empty ratio, but again assigns fewer tasks than *JSprit*.

Table 7.2: Comparison between the solutions obtain by the solvers using 30 problem instances. Bold value represent the best mean value for a KPI. The red and green colors represent the percentage change from *JSprit* for negative and positive criteria for the passengers problem, respectively.

	Jsprit	**OR-Tools**		**OptaPlanner**	
	Mean	**Mean**	**Change**	**Mean**	**Change**
Optimization Time (sec.)	**80.30**	223.23	178%	130.97	63%
Num. Iterations	424	134	—	1923	—
Assigned Ratio	**0.87**	0.84	-3%	0.77	-12%
Working Duration (h)	383	**335**	-12%	357	-7%
Effective Duration (h)	124	143	15%	**111**	-11%
Driving Duration (h)	**98**	107	9%	113	16%
Distance (km)	6773	7037	4%	**4724**	-30%
Distance Empty Ratio (km)	0.37	**0.32**	-13%	0.34	-7%
Payment	**3917**	3636	-7%	2890	-26%
Avg. Speed (km/h)	68	69	—	69	—
Not Assigned Vehicles	4.7	**4.8**	2%	2.8	-39%

7.1.4 Discussion

Despite the results favoring *JSprit*, the aspect of ease of developing new models should not be overlooked. Table 7.3 depicts the trade-offs to implement the passengers model with each of the solvers, providing a take home message to the reader. Lets start by the advantages, the *JSprit* already provides good results without changing the hyper-parameters of the R&R, although it has more than 20 parameters, but to try to match the performance of *JSprit* time was invested in modifying the settings of the remaining solvers without success. For construction heuristic, only *OR-Tools* matched the performance of *JSprit*, but in the majority of the problem instances it was slower. The passengers model can be quickly configured with *OR-Tools* showing the fastest development, but in reality it did not involved any development just the configuration of parameters, whereas *JSprit* and *OptaPlanner* involves weeks of time-effort of a developer to build the passengers model, which in the case of *OptaPlanner* the effort is not worth it considering the results. However, from a software architectural perspective the *OptaPlanner* provides the fastest interface to implement and change constraints using *Drools*, whereas in *JSprit* the process of developing a constraint can take a few days, for instance to implement a constraint to limit the maximum working hours can take at least a day, this time is sufficient to implement a first version of all constraint in *OptaPlanner*.

After more than a month without looking at the models code, the *OptaPlanner* was the most intuitive to understand mostly due to the inversion of control which removes the procedural flow of the code, but this makes the solver more difficult to debug than the others. However, *OptaPlanner* required to develop a domain model, state updaters to implement the requirements of the problem and also custom moves to try to match the performance of the remaining solvers. Moreover, debugging was difficult for being a framework with inversion of control there are several operations in which the user loses control, for instance during the development of the domain model. At last, using *OR-Tools* can be a risk, since if this client wants a new feature, for instance a periodic rest, it cannot be implemented with this solver without change the source code (in C++), and in *JSprit* too, but since the solver is written in Java forking *JSprit* is much simpler, this is what *Sentilant* did to answer some problems. Lastly, *JSprit*'s R&R is composed by two types of soft constraints: insertion and cost function costs, for the reader's knowledge the comfort upper bounds, distance traveled cost and payback are not part of the cost function, but instead of the insertion cost. If a decision between them is not made, it implies having to try both.

Table 7.3: Comparison for the passengers problem regarding advantages and disadvantages of choosing a solver over another for a CVRPPDTW, and considering the ease of developing and maintaining the model.

Jsprit	**OR-Tools**	**OptaPlanner**
Positive • Best overall results • No algorithm's hyper-parameters configuration needed • Fastest construction heuristic while achieving best initial results	**Positive** • Fastest development • Low learning curve	**Positive** • Easy to implement custom soft constraints • Provides a benchmarker to compare different hyper-parameters configurations • Easy to maintain promoting good source code structure and extensibility
Negative • Decide between insertion cost and/or cost function to implement a soft constraint • Hard constraints time-consuming to implement • Highest learning curve • Slowest development if performance is disregarded	**Negative** • High probability of failing to respond to future client's needs of this model	**Negative** • Many time spent choosing the algorithms and configurations to improve performance • No dedicated implementation to lead with pickup and deliveries • Required to develop the domain model • Required to develop custom states updaters • Slowest development (due to custom moves) • Most difficult to debug

Based on these results, a decision was made to choose the solver for the fuel problem. To implement in *OR-Tools* it would be necessary to modify the source code, and in *OptaPlanner* to heavily invest in custom moves, however the current **results favor *JSprit* both in performance and solution quality**, therefore the *Sentilant*'s solver using ***JSprit* was selected to implement the fuel transports model**.

7.2 Fuel Transports Model

First a statistical description of the instance generated to test the model in 7.2.1, results of the performance and quality of the solution 7.2.2 and an overall discussion about the results in 7.2.3.

7.2.1 Data Description

Data about a real working day of the client transporting fuel on 8th August 2020 was obtained, including locations and schedules of all cargo centers and gas stations. The sources of the data were: i) file containing information of 161 gas stations and more than 500 clients that are not gas stations, ii) file containing information about cargo centers, iii) five files describing 57 orders with the identification of the clients and the fuel quantities ordered, and iv) a file describing the tractors, since the fleet is not homogeneous. The remaining information was gathered from conversations with the client. The reader can obtain that information from the problem statement on section 6.2.1.

Resulting testcase has 57 **tasks** and 26 **resources**. Let start by the resources, all tractors, except one, have a 6-tank cistern with a total capacity of 36k liters, but the capacity of the tanks differs. The most common configuration, ending at the rear of the cistern, is: 11k, 5k, 4k, 3k, 7k and 6k. The remaining tractor has a 3-tank cistern with 5.9k, 4.3k and 4.7k (Total: 14.9k) liters capacity. **Loading time** of 1 hour is considered, 10 **maximum working hours** and 8 **driving hours**, and a **rest time** of 45 minutes every 4 hours. 23

out of the 26 tractors have a **work block** [8:00; 19:00]. The remaining resources have a work block for each shift, the **first shift** [00:00; 15:00] and the **second shift** [13:00; 03:00]. The **start and end locations** were not presented in the data, so 15 resources were considered to start in cargo center 1, 7 tractors in cargo center 2, and 4 tractors in cargo center 3. Approximately corresponds to the amount of pickup locations of the tasks.

There are 3 **pickup locations** i.e., cargo centers in this example. For the tasks: 37 (0.65%) in cargo center 1, 9 (0.16%) in cargo center 2, and 11 (0.19%) in cargo center 3. The **delivery locations** corresponds to the clients' gas station. A sufficient large **time-window** was considered for the pickup, the cargo centers are always available. For 22 (0.39%) of the tasks, the gas station does not impose any time restrictions. Eight of the tasks (0.14%) have a time-window between [00:00; 06:00]. The remaining tasks have a time-window with a mean start and end in hours [9:35(SD=1:41); 14:18(SD=2:47)]. It includes the exclusion of interval of time that cannot deliver. There are 6 tasks with **excluded time-windows**, the time gaps are [17:30; 19:00], [11:30; 15:30], [11:30; 15:30], [13:30; 15:30], [12:00; 15:00], and [11:00; 15:00]. The considered **service time** floats between 1 hour, 45 and 30 minutes for cargo above 18k, 7.2k and 0k, respectively. There are 6 types of **products**, 123 products requested with mean 7476.42 (SD=5233.34; min=2000; median=6000, max=30000). Sum **products quantity** of an order as a mean of 16133.33 (SD=7464.04; min=4000.00; median=14800.00; max=38000.00). Moreover, the used **distance cost** is 4 and 1 for **time cost**.

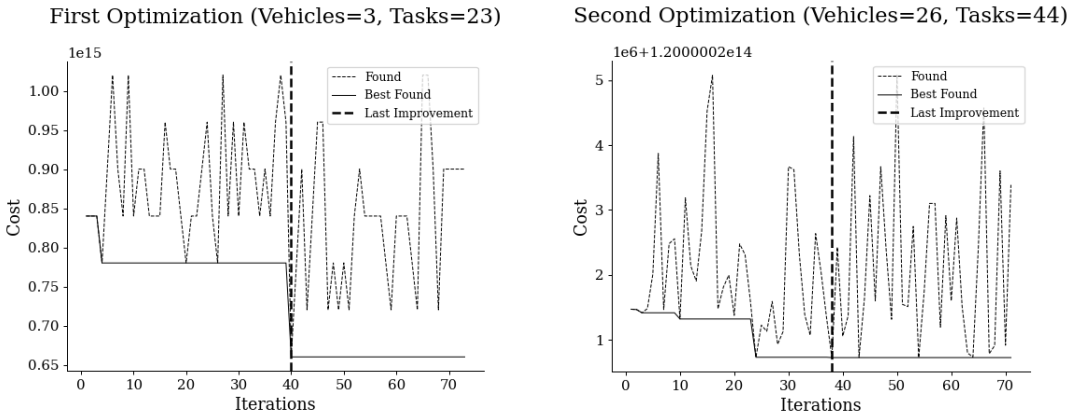
Table 7.4: Percentage of hard-constraints evaluated as broken, thus prevented to insert a task, during the recreate phase of the algorithm for the 1st and 2nd optimization with 47924 and 2580 violations, respectively.

Hard Constraint	Percentage of Broken Hard Constraints	
	1st Shift	2nd Shift & No Shift
FirstPickupTasksAtStartLocationConstraint	(1728) 6.33%	(1804) 3.76%
PreventPickupWhileVehicleIsLoadedConstraint	(16714) **61.27%**	(37040) **77.29%**
ShipmentTankLoadActivityConstraint	(0) 0%	(267) 0.56%
VehicleDependentTimeWindowPeriodicRestConstraint	(8836) 32.39%	(6233) 13.01%
WorkTimeLimitConstraint	(0) 0%	(2569) 5.36%
PreventSubTasksForSameUserConstraint	(0) 0%	(7) 0.01%
DrivingTimeLimitConstraint	(0) 0%	(4) 0.02%
Total	(2580) 100%	(47924) 100%

7.2.2 Results

The number of constraint violations is a metric of reliability of optimization algorithms. Table 7.4 describes the percentage of constraints broken, the bottleneck of the algorithm is *PreventPickupWhileVehicleIsLoadedConstraint* preventing to insert a pickup between two deliveries. This indicates i) if the truck could be loaded at any time, the results would be better, and ii) the algorithm is not prepared to avoid exploring solutions that break this constraint. The second most violated constraint is *VehicleDependentTimeWindowPeriodicRestConstraints* that corresponds to conform with the time windows. The remaining hard constraints represent less than 10% of total violations in each optimization.

An usual metric for measuring the performance of an optimization algorithm is by obtaining the cost of the solution found in each iteration of the algorithm. If the improvement is low or there is no improvement at all, the computing time is considered wasteful. The algorithm was configured to end after 32 iterations without improvement, Figure 7.1 describes the cost of the solution and the best solution found in each iteration.



(a) Last improvement in iteration 41

(b) Last Improvement in iteration 39

Figure 7.1: Cost function values, the algorithm stops after 32 stalled iterations. For the first optimization 12 assigned tasks (0.52%) and 42 assigned tasks (96%) for the second.

Table 7.5: Comparison of KPIs between the initial solution and the best solution found. Green means the solution improved towards the client's criteria, otherwise red.

KPI	Initial Solution	Best Solution Found	Change
Assigned Ratio	0.95	0.96	1.05%
Working Duration (h)	155	147	-5.16%
Effective Duration (h)	135	135	0%
Driving Duration (h)	60	59	-1.67%
Distance (km)	5687	5624	-1.08%
Distance In Empty Ratio	0.43	0.42	-2.33%
Periodic Rest (h)	2	3	50%
Not Assigned Vehicles	4	6	50%
Quantity Transported (liters)	834600	886600	5.87%
Free Space (liters)	223800	192900	-13.81%
Free Space Ratio	0.27	0.22	-18.52%
Cisterns	30	30	0%
Used Compartments	153	162	5.88%
Unused Comp. Ratio	0.16	0.11	-31.25%

KPIs for the best solution found are compared with the initial solution from the algorithm in Figure 7.5. The solution represents the merged results from both optimizations. Comparing the initial solution with the best solution there is no significant improvement, the number of assigned tasks and the total distance are very close, less than 2% difference in both cases. The algorithm was also ran in increasing base exponents of 2, b^2, up to 4096 iterations and no significant improvement was found i.e., more task assigned.

The client does not appreciated some generated routes as depicted in section 6.2.3 of the previous chapter. Routes unappreciated by the client were classified as: i) large distance between deliveries, ii) delivered first to a further away gas station, iii) transits through Ponte 25 de Abril where the hazardous transports should not pass. Based on the classification, **routes were ranked as bad, acceptable, and good**. A route with bad ranking is classified as i), ii) or iii); with acceptable ranking as only one delivery or a moderate distance between deliveries (this is subjective); otherwise, ranked as good.

Table 7.6 illustrates the results for 5 different configurations of the solver. The scenarios with the soft constraints avoid *Ponte 25 de Abril* and use the improved distribution algorithm. The *Deliver Closest First* scenario seems to provide the best results, but the majority of the bad and neutral routes in this scenario are due to a large distances travelled

Table 7.6: Classification of the routes appreciated by the client with different configuration. Unique Delivery is represented by 2 values, the first is the number of routes with just one delivery and the second is the number of single deliveries.

Scenarios	Routes	Distance (km)	Unique Delivery	Bad	Acceptable	Good
Second Delivered Report	28	6170	9 / 15	2 (7%)	**13 (46%)**	13 (46%)
Avoid Ponte 25 Abril and Improved Distribution Algorithm	23	5690	2 / 11	3 (13%)	10 (44%)	10 (44%)
Deliver Closest First	23	5772	2 / 9	2 (9%)	4 (17%)	**17 (74%)**
Avoid Travelling Large Distances Between Simultaneous Deliveries	24	5630	5 / 12	3 (13%)	8 (33%)	13 (54%)
Both constraints (Final)	23	5624	4 / 10	2 (9%)	7 (30%)	14 (61%)

between simultaneous deliveries. When both constraints are combined, the results improve toward the client's preferences, but 2 bad routes were generated. An analysis to those bad routes revealed that due to time-windows constraints a better route could not be formed. For instance, the driver goes to A then B, but going first to B than A is not possible.

7.2.3 Discussion

The results presented in this section aim to provide a view about the performance of the last version of the model, but most of the work on this model focused on its development and in satisfying customer preferences. Regarding customer preferences, the classification of the routes and comparison of the results, between different work plans, was extremely important to find a way to improve the solution toward the client's needs. Some of the days, more time was invested in analyzing the current solution at that moment, than implementing something. Little time was invested in research for this model, instead it was invested in understanding the *JSprit* algorithm, but there is not much information on the internet regarding the R&R, and about our requirements implemented in any of the compared solvers.

Some requirements were more difficult to implement than others, the most difficult to implement were the tanks states, periodic rest, improve distribution of the load and limit working time limit. For instance, to improve the performance, instead of update the tank based on the previous states, each set of sequential pickups was updated at once. The same concept of sequential pickups (and deliveries) applied in the evaluation of the constraints made us achieve in some cases better results. One the other side, some functionalities were straightforward to implement, for instance the solver supports multiple time-windows, so to implement a way to excluded time-windows this functionality was simply used.

7.3 Planning System API

Two results are presented about the planning system, the processing time of a synchronous request and a brief description of the API and its documentation.

Tests to the planning system were made with the problem instance generated for the fuel problem. In Table 7.7 the mean time spent by the system to process a synchronous request. As expected, the first optimization is faster because the search space is smaller, half of the

tasks to assign and there is only 3 resources to allocate them, compared to 26 in the second optimization. Moreover, the solver ran in single-thread and occupied around 170 megabytes of RAM for the first optimization and 190 for the second in the Java Virtual Machine.

Table 7.7: Time, in seconds, to process a synchronous request. Processing Time: time took by the solver from receiving the problem instance until sending to the queue; Pipeline: same as processing but for an optimization; Valhalla: request and receive time and distances; Search Time: optimization algorithm run-time; (1) and (2) means 1st and 2nd optimization; Tested with 10 requests.

	Mean	Std
Pipeline (1)	2.71	0.52
Valhalla (1)	2.10	0.17
Search Time (1)	0.50	0.41
Pipeline (2)	9.30	0.58
Valhalla (2)	4.44	0.14
Search Time (2)	4.63	0.55
Processing Time	12.20	1.24
Sync Request Time	12.49	1.24

The API was documented using *Swagger* and can be accessed through the planning system in the endpoint */api/api-doc/*. The documentation page is illustrated in Figure 7.2 with two endpoints. The endpoint */api/plan* exposes an HTTP post method to submit a problem, and a get method to retrieve a plan given its identification. The system only interprets messages in a JSON-format, and the API has 24 objects documented.

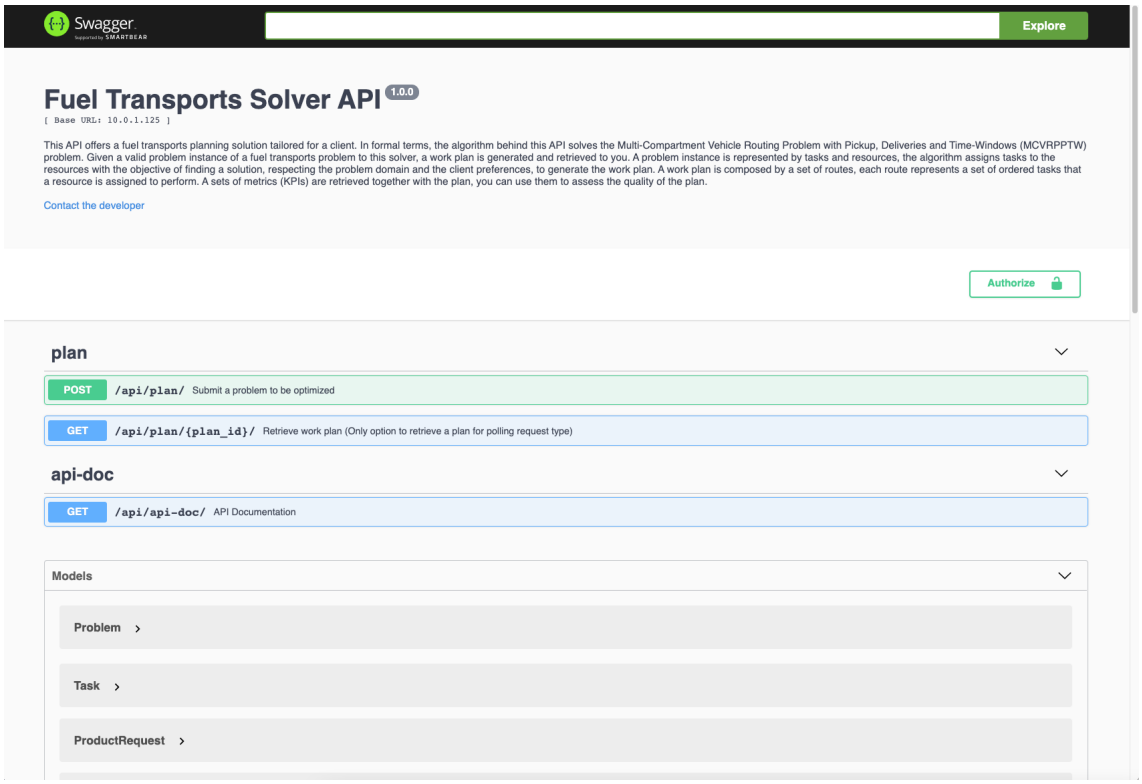


Figure 7.2: API Documentation: Top of the page.

To submit a problem, the API requires a valid authentication token i.e., key, a valid problem instance and to choose the type of request synchronous, asynchronous or webhooks.

If the type is webhooks, an callback url has to be provided to the server to send the response, this part of the documentation is depicted in Figure 7.3. Four response codes are implemented: success, bad request error, unauthorized error and unprocessable entity. The fourth response code means the problem instance is not valid, or there is missing information required to the solver generate the work plan.

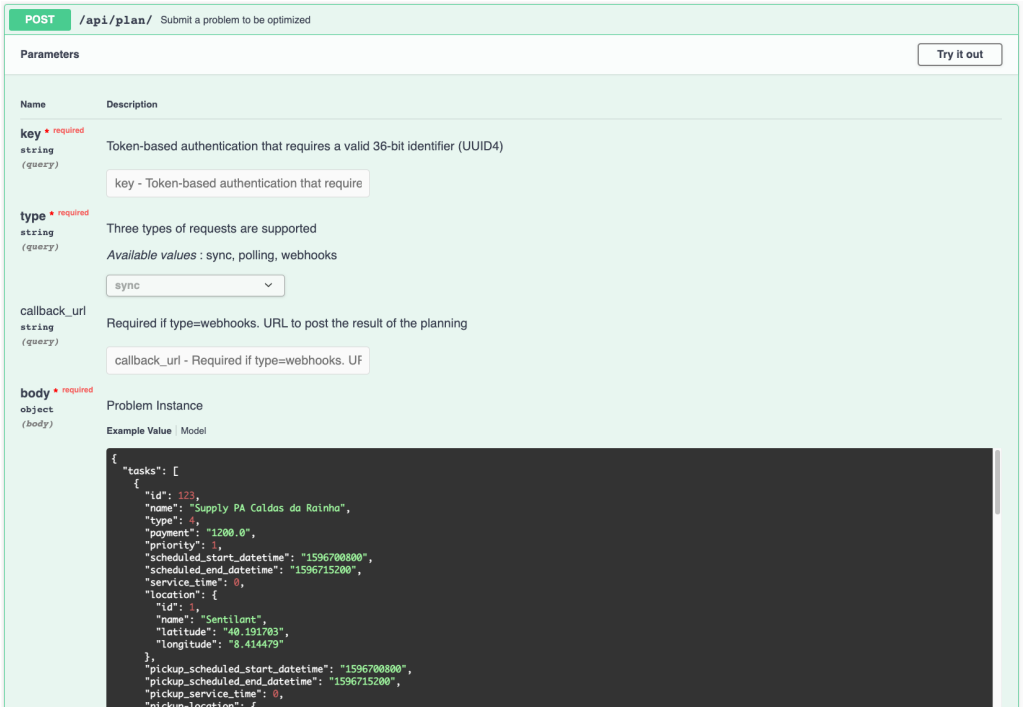


Figure 7.3: API Documentation: Submit a problem instance to generate a work plan.

7.4 Feedback Report

To present the optimization results to the client, a PDF report is generated from the problem data and the resulting work plan obtained from the solver represented in JSON-format. The PDF **report is divided into three parts**, the first part presents two tables with information about the drivers and the tasks, the second part presents the routes assigned to drivers and a Gantt with the schedules of the drivers.

The beginning of each table, in the **first part** is represented in Figures 7.4 and 7.5, describing the problem instance from section 7.2.1 in a table format understandable by the client. The first part was crucial in the first feedback, since several problems were detected (e.g., considered time required to load a cistern, schedule of the shifts), the illustration of this information generated discussion with the client important to collect accurate data about the problem.

id	Nome	Tanques	Carregar	Limite	Descanso	Turno 1	Turno 2	Início	Fim	
0	L-143225	['11000', '7000', '3000', '6000', '4000', '5000']	01:00	Condução: 08:00 Trabalho: 10:00	04:00 00:45	06/Aug 00:00 06/Aug 15:00	06/Aug 13:00 07/Aug 03:00	CC - GALP Aveiras (39.17025,-8.92521)	CC - GALP Aveiras (39.17025,-8.92521)	X
1	L-147889	['11000', '7000', '3000', '6000', '4000', '5000']	01:00	Condução: 08:00 Trabalho: 10:00	04:00 00:45	06/Aug 00:00 06/Aug 15:00	06/Aug 13:00 07/Aug 03:00	CC - GALP Aveiras (39.17025,-8.92521)	CC - GALP Aveiras (39.17025,-8.92521)	X

Figure 7.4: Start of the table describing the resources.

The **second part** presents the work plan information depicted in Figure 7.6: the routes presented in a table format, metrics, and a map showing the geographic locations to visit,

id	TOP D.	TOP 95	GAS.95	GAS.98	G.ROD.	G.AGR.	Pri.	Serviço	Local	Janela Temporal	Exclusões	
0	17000		2000		5000		1	00:00 01:00	De: CC - GALP Sines (37.964556,-8.798197) Para: PA FARO (37.03709,-7.95719)	06/Aug [00:00;03:00] 06/Aug [00:00;06:00]		X
1	7000		2000				1	00:00 00:45	De: CC - GALP Sines (37.964556,-8.798197) Para: PA TAVIRA (37.12992,-7.638830)	06/Aug [00:00;03:00] 06/Aug [00:00;00:00]	[17:30;19:00]	X

Figure 7.5: Start of the table describing the tasks.

for each of the routes. In the first version of the report, the routes were not graphically presented, but the client requested to show the routes and to consider a set of sequential pickups a load. The client was interested in the last column that presents the occupation of the cistern, conveying this to be an important metric, as described in the PE of the tanks states in section 6.2.3. The client explained that minimizing the distance traveled and maximizing the quantity loaded in the cistern were the most important objective, however, the client also wanted as many tasks as possible to be assigned.

The **third part** shows KPIs, and a illustrative Gantt diagram in Figure 7.7, representing the schedule of the drivers. The Gantt represents the results from section 6.2, but the important thing to convey is how results were presented to obtain feedback. When the Gantt was presented to the client for the first time, it involved a long discussion about time-related requirements, and the client suggest to implement a visualization like this in the FSM service, but this is out of scope of this work.

ID: 1 Veículo: L-147889 Turno: 1 [00:00;10:01]

Id	Local	Janela	Chegada	Partida	Duração	Dist.	Tanque 1	Tanque 2	Tanque 3	Tanque 4	Tanque 5	Tanque 6	Ocup.
I	CC - GALP Aveiras			00:00	00:00	0.0	**11000**	**7000**	**3000**	**6000**	**4000**	**5000**	0%
	CC - GALP Aveiras	[00:00; 03:00]	00:00	01:00	00:28	48.2	11000 TOP D.	7000 TOP D.	1000 TOP D.	6000 TOP D.	3000 G.ROD.	5000 G.ROD.	91% (91%)
21	PA Vialonga	[00:00; 06:00]	01:28	02:28	00:28	48.1	0 TOP D.	0 TOP D.	0 TOP D.	0 TOP D.	0 G.ROD.	0 G.ROD.	0% (-91%)
	CC - GALP Aveiras	[00:00; 03:00]	02:56	03:56	00:39	64.3	10000 TOP D.	5900 TOP D.	200 G.ROD.	5000 TOP 95	4000 G.ROD.	4700 TOP 95	82% (82%)
23	PA FAMOES	[00:00; 06:00]	04:36	05:21	00:14	12.8	0 TOP D.	5900 TOP D.	200 G.ROD.	0 TOP 95	4000 G.ROD.	4700 TOP 95	41% (-41%)
27	PA Lisboa Av. Roma	[00:00; 06:00]	05:35	06:20	00:35	59.1	0	0 TOP D.	0 G.ROD.	0	0 G.ROD.	0 TOP 95	0% (-41%)
	CC - GALP Aveiras	[00:00; 03:00]	06:55	07:55	00:32	52.3	11000 TOP D.	7000 TOP D.	0	5000 G.ROD.	0	2000 TOP D.	69% (69%)
6	CD581 RAPOSA [LUBRIF	[08:00; 12:00]	08:28	09:28	00:32	52.0	0 TOP D.	0 TOP D.	0	0 G.ROD.	0	0 TOP D.	0% (-69%)
F	CC - GALP Aveiras		10:01										

total_assigned_tasks	4	**Distância (Km)**	336.88	**Distância Em Vazio (Km)**	159.28	**Distância em Vazio (%)**	47%
Duração de Trabalho (h)	10.01	**Duração de Serviço (h)**	06.30	**Duração de Condução (h)**	03.31	**Duração de Rota (h)**	10.01
Tempo de Espera (h)	00:00	**Duração Efetiva (h)**	10:01	**Custo (0.93 por Km)**	313.3	**Quantidade Transportada**	87800
Espaço Não Ocupado	20200	**Cisternas**	3	**Tanques Utilizados**	16	**Tanques Não Utilizados**	2

Figure 7.6: Example of the presentation of a route in the report with the work plan information. In the map, the green node is a cargo center and the black nodes are the gas stations to deliver the fuel, the deliveries are numerated.

As a **reflection**, there are many ways to do software demos, in this case, to present results of the development of an optimization model for a planning problem. Initially, a single web page would be built with the results, but too much time would be invested. As alternative, a PDF was generated with a script developed in a couple of days, with the advantage of reducing the ways to convey information to: tables, figures or text without hidden information e.g., requires to click. Also, the client could print reports to compare different versions of the work plans, ensuring no deformation in the document.

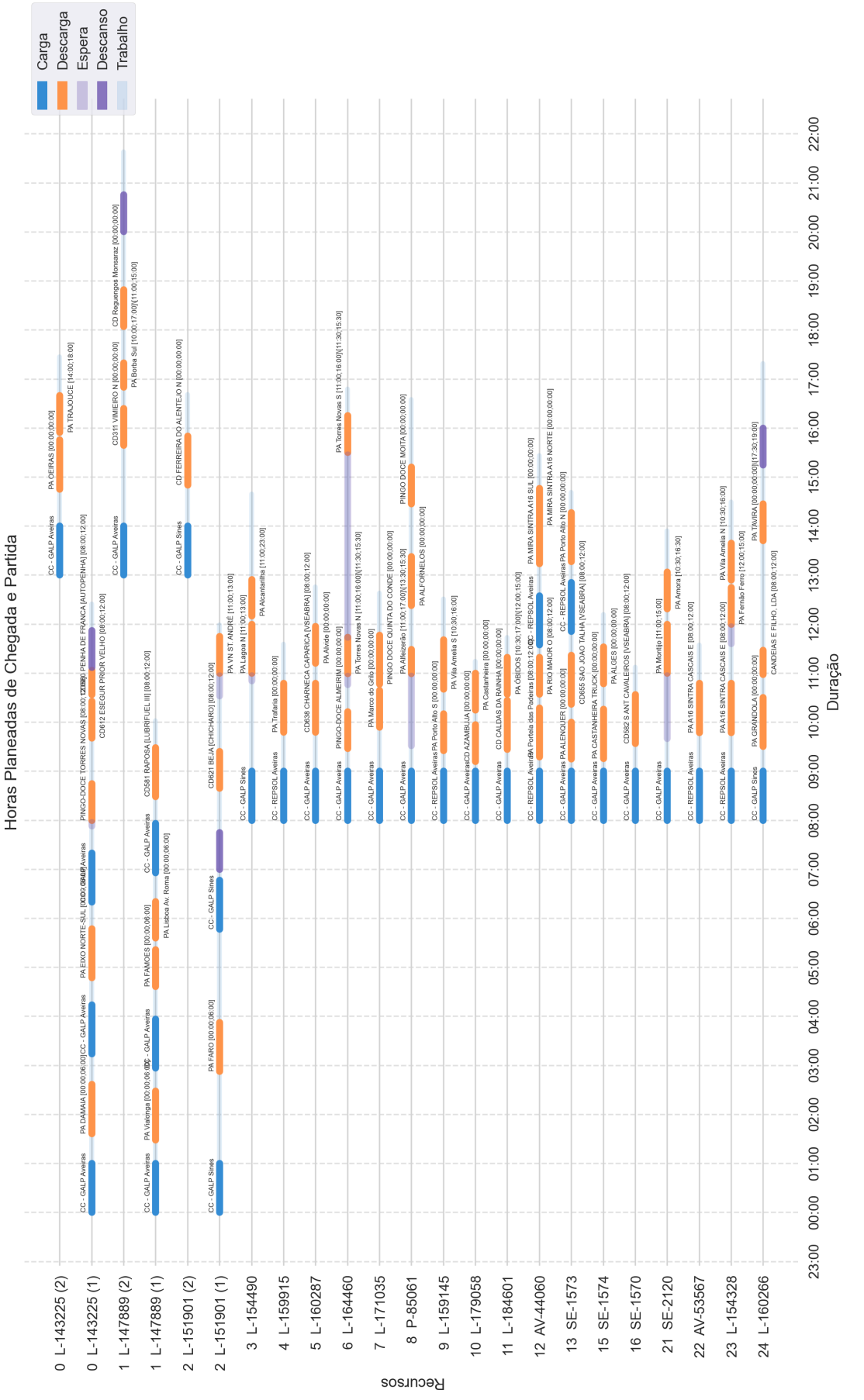


Figure 7.7: Gantt chart representing the routes timeline. The labels are written in Portuguese.

7.5 Planning System Integration

Sentilant provides an FSM service called *DrivianTasks*. In the case of the fuel model client, the service has been tailored to his business needs, and currently does not use automatic planning for any of his problems. In this section, the flow of the client from importing orders to the visualization of the resulting work plan is demonstrated. The performed integration corresponded in adapting the existing optimization user interface, in the base version of *DrivianTasks*, and integrate the communication with the planning system. A request for optimization involves retrieving data from the database, build the request, send the request to the API, read the response and writing the modifications in the database.

The **first step** of the client is importing orders to the service in the interface depicted in Figure 7.8. Usually, involves importing more than one file, each file corresponds to orders that are loaded in a specific CC.

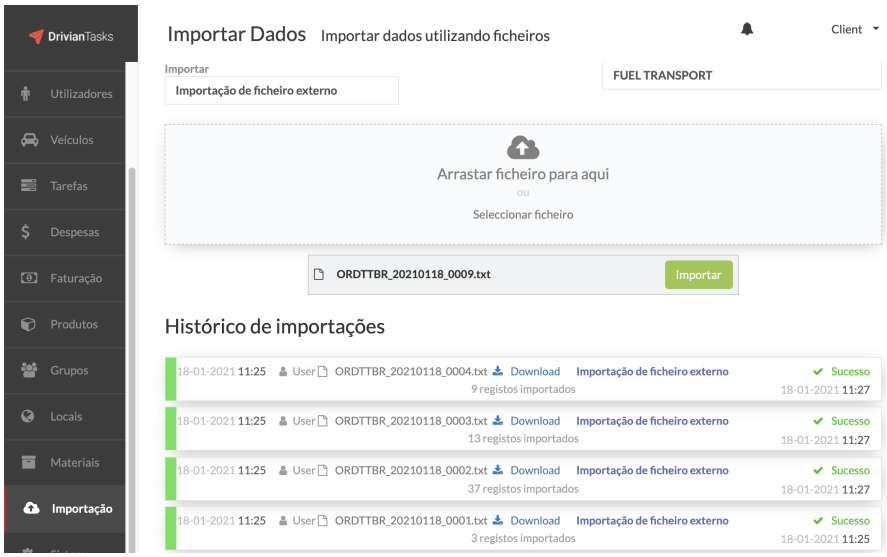


Figure 7.8: *DrivianTasks*: Interface to import data such as the fuel transportation orders.

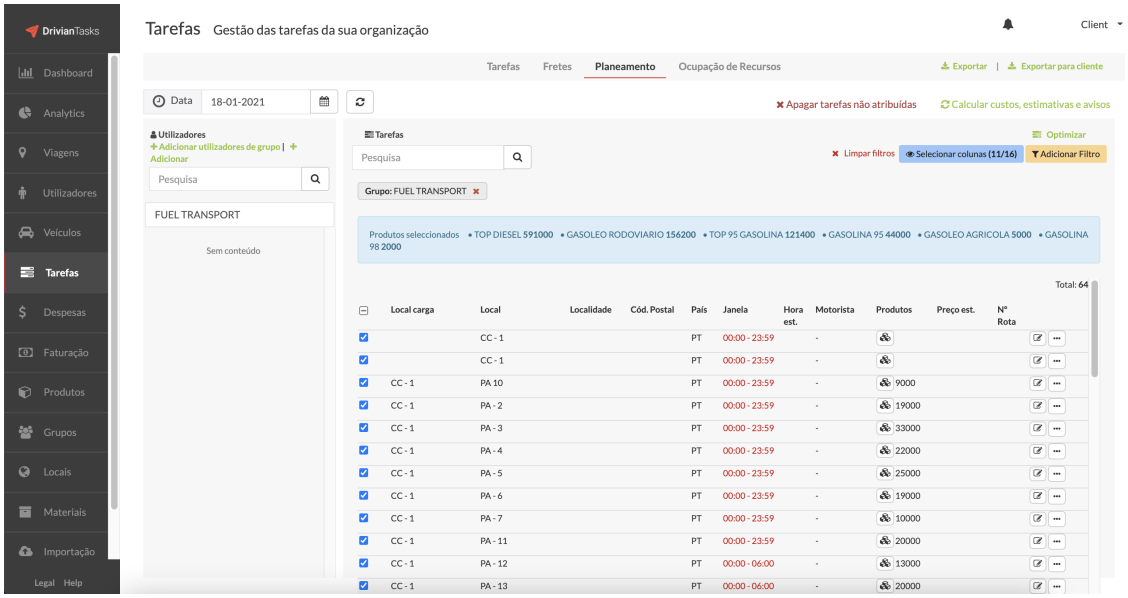


Figure 7.9: *DrivianTasks*: Planning panel where tasks can visualized and edited.

In the **second step**, group and day of the planning has to be chosen. The client does several types of transports and uses a concept of groups, part of his drivers, vehicles and the imported tasks in the previous step are associated to a group. Then, the tasks to plan can be selected as shown in Figure 7.9, in this case all tasks were selected.

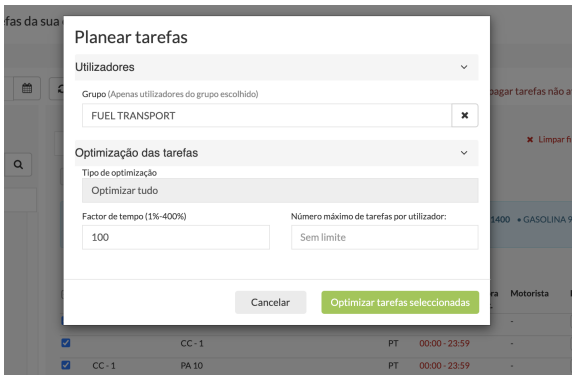


Figure 7.10: *DrivianTasks*: Planning panel to configure additional parameters before optimize the selected tasks.

In the **third step** after clicking the optimize button in the top right corner of the tasks panel, a windows to configure parameterizations is displayed. For this client, this button was not active, it was activated and almost all existing configurations were removed from the panel, except speed factor and maximum number of tasks per user. When the green button in Figure 7.10 is clicked to optimize, the integration done in this work comes in, that reads the required data and saves the work plan. During the process the client has to await for a response.

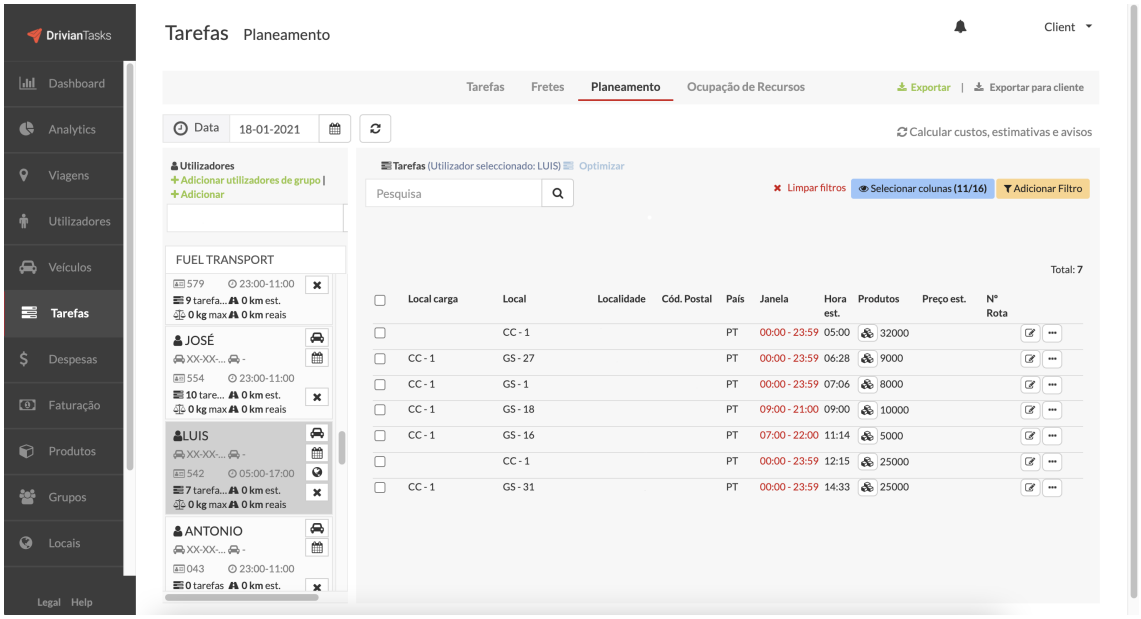


Figure 7.11: *DrivianTasks*: Visualization of a work plan for a driver with 7 tasks assigned after the optimization. On the right side of the time-window in red color is presented the expected start time of the task obtain from the solver.

After the planning process, the client can visualize the schedule for the drivers, can make any modification to the plan or even re-optimize the plan. In Figure 7.11 is described the work plan for a driver with 7 assigned tasks. The load in a cistern can be visualized in Figure 7.12, the distribution of cargo from the solution obtained from the solver for a load

(i.e., in VRPPD convention 2 pickups) with 2 deliveries.

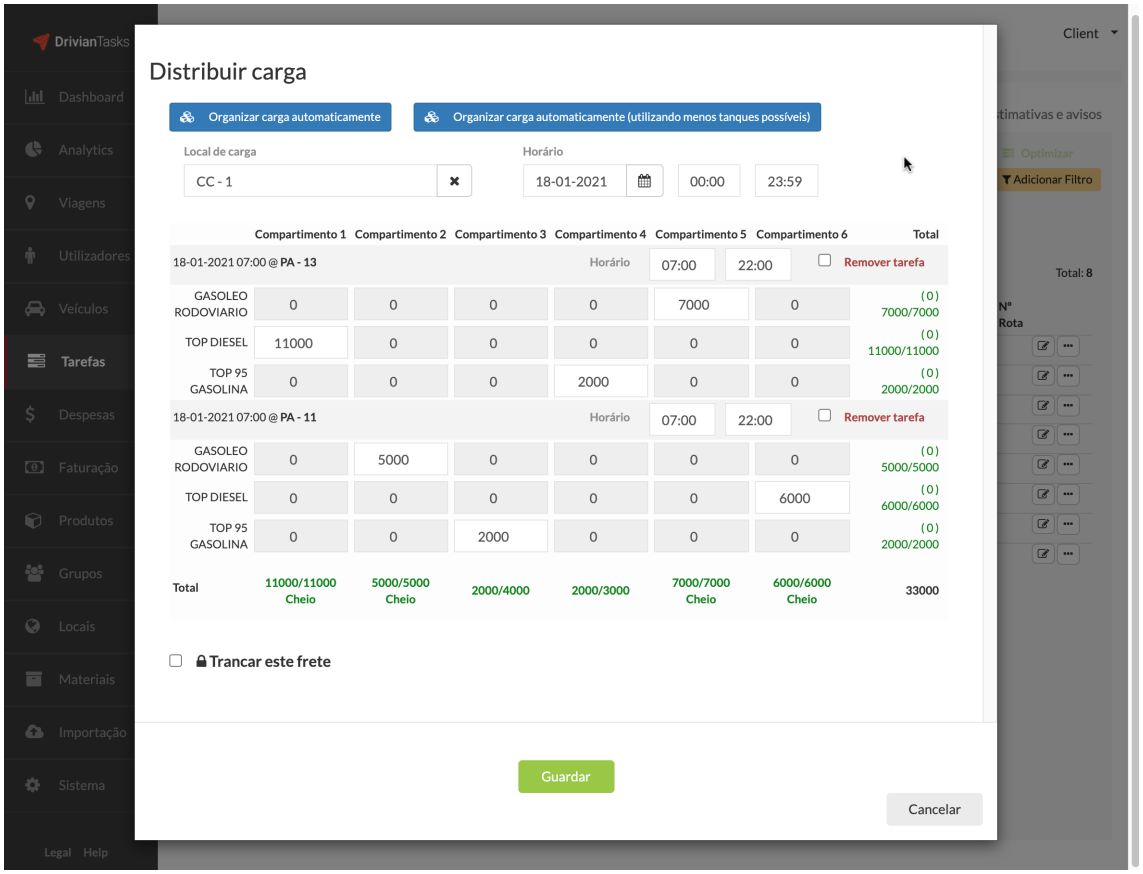


Figure 7.12: *DrivianTasks*: The loading of the cistern can be visualized.

At this stage, the results were validated manually without integration and unit tests, although the planning can be demonstrated to the client. This last part of the work illustrates the real-world application of the developed optimization model, without this part, regardless of the results of the optimizations and the technical difficulty, this work would seem incomplete as the way it was presented.

Chapter 8

Conclusions

The initial proposal involved using *OptaPlanner* to answer 3 optimization problems. By the end of the first semester, *Sentilant* proposed to develop a model to answer a fuel transport problem for a real client, redirecting this work to answer that problem with inputs and feedback from the client.

As a contribution to the *Sentilant*'s FSM system, a planning system API was developed and deployed with Docker on a remote server, allowing external software systems to obtain work plans including aggregated KPIs for the fuel transport problem.

This project involves two areas of expertise. The **mathematical optimization** that involves understanding optimization algorithms to implement functionalities in the problem layer of a solver, thus satisfying the business requirements of clients. For the **software engineering** involves requirement gathering, documentation, and the *Agile* software development process followed to manage the project.

For the **optimization** part, in the first semester, an extensive review of 3 state-of-the-art VRP solvers targeted to solve real world scenarios was performed. In addition to a conceptual analysis, the passengers model was implemented, and compared with 30 real test cases, concluding that *JSprit* can allocate more tasks in less time. In the second semester, *JSprit* was chosen to implement the fuel transport model for a client of the *Sentilant*'s FSM system. The solver was extended with a set of features justified with PE, obtained from the client's feedback. Moreover, feedback was crucial to refine the problem instance used to test the model, so confidence in the results shown in 8.2 can be given.

For the **engineering** part, in the first semester, the ASRs were defined, consisting of functional requirements, QAs, and technical and business constraints, also risks were identified and mitigated. Building the utility tree in 4.12 was an important parts of this work, in just one table is given a view of the main requirements that the system should satisfy. The development process started in the second semester with 6 sprints, lasting 2 weeks each. Four sprints were dedicated to the **fuel model**, and 2 to the **planning system**. Providing an answer to the fuel problem by the end of December 2020 was the threshold of success for this project, thus more time was invested on the model and results visualization than on the system. After developing the API, two additional sprints were followed to integrate the solution in the FSM service used by the client to demonstrate the planning proposal.

8.1 Overall Conclusions

All the goals set for this work were accomplished. Despite the routing solvers offering implemented optimization algorithms, developing a model takes time and above all, the student learned that there are real-world requirements for the models that are not part of the optimization problem, such as avoiding locations, obtaining real distances and durations depending on the client needs, and multi-optimizations to lead with shifts.

This was the student's first contact with optimization algorithms and there were difficulties in how to expose and explain the results. Technical decisions made in terms of optimization were taken by the student, and in the eyes of experts may not be the best, namely in the adopted optimization language to explain the implementation, and in the application of preferences elicitation.

Choosing the solver was the most difficult decision in this work, fortunately all model's requirements were successfully implemented. This shows that investing time in comparing technologies before hand, namely solvers, can decrease the likelihood of failure to respond to the client, and also that the quality of the work plans offered be better.

It must not be forgotten that *Sentilant* has a need to reduce the time invested in the development of its models, although in this work a better alternative has not been found, so confidence has been given in *JSprit*.

To conclude, this project was a great challenge that involved making technical and management decisions during the course of the project, contributing to the enrichment of hard and soft skills that helped the student to become better at a professional level.

8.2 Future Work

Regarding the integration with the FSM service, for now the parameterization is hard-coded, but in the future a better interface should be developed specifically for this client's problem. For instance, allowing the client to change the loading time of a cistern, and specify locations that should be avoided, as bridges.

As future work to improve the fuel model is proposed to: i) generate more instances to test the model and improve soft constraints e.g., delivery closest first, ii) each order has one respective CC to load the fuels, so let the solver optionally accepts a set of possible CCs for each order and assigning the most adequate, and iii) tractors transporting hazardous materials must, whenever possible, travel on motorways, thereby providing a new KPI with the toll cost of the work plan.

Moreover, currently there are clients of the FSM system interested to obtained work plans that *Sentilant* as no answer, so in the future new tailored models will be developed and integrated into the planning system.

References

Bass, L., Clements, P., and Kazman, R. (2003). Software Architecture in Practice , Second Edition. *Software Architecture*, page 560.

Borges, H., Hora, A., and Valente, M. T. (2017). Understanding the factors that impact the popularity of GitHub repositories. Institute of Electrical and Electronics Engineers Inc.

Burke, E. K. and Bykov, Y. (2008). A late acceptance strategy in hill-climbing for exam timetabling problems. *7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008*, (January 2008).

Gendreau, M. (2006). An Introduction to Tabu Search. *Handbook of Metaheuristics*, pages 37–54.

Groër, C., Golden, B., and Wasil, E. (2010). A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation*, 2(2):79–101.

Holborn, P. L., Thompson, J. M., and Lewis, R. (2012). Combining heuristic and exact methods to solve the vehicle routing problem with pickups, deliveries and time windows. *Lecture Notes in Computer Science*, 7245 LNCS(2):63–74.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34(8):2403–2435.

Proctor, M. (2012). Drools: A Rule Engine for Complex Event Processing. pages 2–2. Springer, Berlin, Heidelberg.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., and Dueck, G. (2000). Record Breaking Optimization Results Using the Ruin and Recreate Principle. *Journal of Computational Physics*, 159(2):139–171.

Simon Brown (2011). The C4 model for visualising software architecture, https://c4model.com, (Accessed: 2020-06-18).

Surana, P. (2019). *Benchmarking Optimization Algorithms for Capacitated Vehicle Routing Problems*. PhD thesis.

Toth, P. and Vigo, D. (2002). 1. An Overview of Vehicle Routing Problems. In *The Vehicle Routing Problem*, pages 1–26. Society for Industrial and Applied Mathematics.

Toth, P., Vigo, D., and Toth, P. (2014). *Vehicle Routing*.

Voudouris, C. (1998). Guided Local Search — an illustrative example in function optimisation. 16(3).

Appendices

Appendix A: Results for the Airport Problem

Table 1: Results for the implementation with *JSprit* from *Sentilant* for the airport problem. Tested with 30 instances randomly taken from the requests of the client between 2018 and 2020.

Id	Tasks	Vehicles	Duration (sec.)	Iter.	Assigned Ratio	Working Duration (h)	Effective Duration (h)	Driving Duration (h)	Distance (Km)	Distance Empty Ratio	Payment	Avg. Speed (Km/h)	Unassigned Vehicles
1	**66**	**13**	10	500	0.74	119	66	55	3408	0.31	1918	67	0
2	**53**	**17**	11	500	0.89	246	61	50	3089	0.32	1960	69	0
3	**78**	**33**	18	500	1	382	99	81	5609	0.29	3460	67	8
4	**14**	**15**	2	201	0.86	74	20	18	1353	0.46	587	74	9
5	**56**	**31**	8	218	1	301	66	53	3562	0.28	2198	66	10
6	**71**	**22**	12	500	1	299	90	72	5064	0.29	3108	68	1
7	**32**	**23**	7	452	1	183	38	32	2249	0.25	1432	68	9
8	**36**	**3**	4	333	0.47	56	11	7	481	0.11	85	67	0
9	**32**	**23**	7	500	1	173	41	32	2230	0.25	1432	67	9
10	**67**	**23**	12	500	0.99	307	87	70	4747	0.31	2844	67	1
11	**52**	**23**	13	500	0.98	322	73	61	4236	0.36	2242	69	0
12	**65**	**22**	7	267	0.92	260	75	61	4096	0.33	2381	67	0
13	**116**	**16**	20	248	0.58	198	86	70	4697	0.37	2393	67	0
14	**110**	**20**	30	500	0.78	288	122	101	7029	0.39	3481	70	0
15	**30**	**15**	5	308	0.83	112	40	35	2616	0.47	1147	70	4
16	**58**	**10**	11	500	0.74	159	56	47	3195	0.39	1699	68	0
17	**95**	**38**	27	500	1	365	121	103	6953	0.42	3862	67	11
18	**83**	**22**	12	333	0.92	283	104	88	6165	0.4	3156	69	0
19	**35**	**1**	1	220	0.17	21	6	4	261	0.66	165	58	0
20	**321**	**58**	291	500	0.94	819	427	352	25236	0.74	14055	71	0
21	**220**	**48**	117	500	0.93	640	268	218	14969	0.73	8433	68	0
22	**54**	**7**	7	500	0.85	640	38	26	1778	0.15	230	69	0
23	**198**	**43**	84	423	0.99	613	240	194	13253	0.77	7965	68	3
24	**205**	**41**	95	500	0.94	546	247	209	14425	0.69	7825	69	0
25	**194**	**37**	105	500	0.85	473	120	82	5602	0.28	6576	68	1
26	**327**	**55**	283	456	0.9	744	246	184	12758	0.18	12393	68	0
27	**321**	**54**	231	411	0.93	733	232	182	12707	0.18	12837	69	0
28	**132**	**38**	44	500	0.97	463	114	79	5331	0.3	4734	65	2
29	**61**	**23**	15	500	1	273	57	41	2850	0.27	305	68	3
30	**565**	**70**	920	338	0.93	1385	467	327	23235	0.14	2620	71	69
Mean	125	28	80	424	0.87	383	124	98	6773	0.37	3917	68	5

Table 2: Results for the implementation done in this work using *OR-Tools* for the airport problem. Tested with 30 instances randomly taken from the requests of the client between 2018 and 2020.

Id	Tasks	Vehicles	Duration (sec.)	Iter.	Assigned Ratio	Working Duration (h)	Effective Duration (h)	Driving Duration (h)	Distance (Km)	Distance Empty Ratio	Payment	Avg. Speed (Km/h)	Unassigned Vehicles
1	**66**	**13**	23	45	0.71	114	58	49	3252	0.34	1824	67	0
2	**53**	**17**	13	36	0.83	218	57	49	3406	0.38	1147	72	0
3	**78**	**33**	68	118	1	359	97	81	5669	0.3	3460	73	7
4	**14**	**15**	1	28	0.86	52	19	17	1310	0.44	587	82	10
5	**56**	**31**	25	63	1	293	64	54	3685	0.3	2198	73	10
6	**71**	**22**	72	184	0.97	284	85	69	4903	0.29	3007	73	1
7	**32**	**23**	8	85	1	166	35	34	2341	0.28	1432	73	9
8	**36**	**3**	7	50	0.42	54	19	14	939	0.38	75	63	0
9	**32**	**23**	8	65	1	149	41	34	2393	0.3	1432	74	9
10	**67**	**23**	54	157	0.97	301	73	59	3926	0.21	2672	69	1
11	**52**	**23**	31	107	0.94	303	69	60	4217	0.39	2118	74	0
12	**65**	**22**	33	97	0.91	271	67	54	3603	0.28	271	69	2
13	**116**	**16**	261	144	0.59	181	84	67	4521	0.25	2771	68	0
14	**110**	**20**	187	87	0.64	247	87	71	4886	0.31	2764	70	0
15	**30**	**15**	42	405	0.83	110	37	34	2491	0.44	1147	79	5
16	**58**	**10**	8	43	0.76	148	57	49	3327	0.41	1802	70	0
17	**95**	**38**	113	108	0.97	334	116	99	6814	0.42	3818	70	13
18	**83**	**22**	136	185	0.9	290	95	79	5493	0.37	3043	70	1
19	**35**	**1**	2	34	0.2	18	8	6	324	0.37	212	50	0
20	**321**	**58**	690	235	0.86	715	355	288	18588	0.3	12582	65	0
21	**220**	**48**	509	175	0.9	565	306	188	12223	0.29	8118	65	0
22	**54**	**7**	14	39	0.8	127	331	42	2924	0.33	215	67	0
23	**198**	**43**	125	199	0.99	585	219	171	11101	0.23	7960	64	5
24	**205**	**41**	205	127	0.89	481	223	187	12140	0.35	7359	65	1
25	**194**	**37**	437	142	0.73	407	186	151	9837	0.34	6185	65	5
26	**327**	**55**	766	175	0.85	669	338	279	17388	0.28	11798	62	1
27	**321**	**54**	717	187	0.87	674	320	273	17462	0.28	11668	64	0
28	**132**	**38**	337	155	0.93	425	143	113	7521	0.26	4552	68	3
29	**61**	**23**	20	96	1	237	70	58	4026	0.26	305	70	4
30	**565**	**70**	1785	440	0.9	1272	619	468	30399	0.25	2555	63	56
Mean	125	28	223	134	0.84	335	143	107	7037	0.32	3636	69	5

Table 3: Results for the implementation done in this work using *OptaPlanner* for the airport problem. Tested with 30 instances randomly taken from the requests of the client between 2018 and 2020.

Id	Tasks	Vehicles	Duration (sec.)	Iter.	Assigned Ratio	Working Duration (h)	Effective Duration (h)	Driving Duration (h)	Distance (Km)	Distance Empty Ratio	Payment	Avg. Speed (Km/h)	Unassigned Vehicles
1	**66**	**13**	32	498	0.68	119	57	47	3081	0.38	1640	66	0
2	**53**	**17**	34	403	0.81	237	60	51	3501	0.41	1820	69	0
3	**78**	**33**	60	1000	1	394	102	83	5708	0.3	3457	69	7
4	**14**	**15**	15	161	0.86	67	20	17	540	0.42	146	75	11
5	**56**	**31**	47	842	1	267	67	54	3523	0.27	2194	66	12
6	**71**	**22**	76	1044	0.97	304	78	61	4115	0.21	2808	67	1
7	**32**	**23**	25	349	1	153	42	37	2625	0.36	1429	71	11
8	**36**	**3**	20	277	0.44	50	19	15	1018	0.33	80	67	0
9	**32**	**23**	35	456	1	123	48	39	2747	0.39	1429	70	10
10	**67**	**23**	47	901	0.92	324	74	59	3839	0.25	2455	65	1
11	**52**	**23**	87	1008	0.98	323	72	60	4145	0.34	2193	69	0
12	**65**	**22**	82	1205	0.91	267	72	57	3826	0.28	2391	67	0
13	**116**	**16**	76	2587	0.51	185	73	59	3852	0.29	2237	66	0
14	**110**	**20**	66	2582	0.59	248	82	66	4422	0.31	2566	67	0
15	**30**	**15**	15	250	0.93	136	45	39	2896	0.46	1297	74	5
16	**58**	**10**	27	453	0.64	140	47	39	2655	0.38	1456	68	0
17	**95**	**38**	137	9748	1	430	129	110	7466	0.46	3853	68	8
18	**83**	**22**	51	745	0.82	290	91	866	236	0.44	2633	68	0
19	**35**	**1**	9	131	0.08	18	6	5	307	0.38	176	65	0
20	**321**	**58**	321	5415	0.68	750	270	217	9726	0.3	8689	45	0
21	**220**	**48**	213	2662	0.81	614	213	169	10992	0.31	6979	65	0
22	**54**	**7**	56	674	0.64	137	36	23	3655	0.32	255	162	0
23	**198**	**43**	93	3000	0.88	671	203	162	10547	0.32	6637	65	4
24	**205**	**41**	187	4048	0.81	539	192	159	10339	0.35	6228	65	0
25	**194**	**37**	265	4576	0.34	475	176	141	254	0.34	5672	64	1
26	**327**	**55**	86	2169	0.46	723	190	87	171	0.4	5597	65	1
27	**321**	**54**	49	1176	0.44	523	128	114	7391	0.51	3088	65	8
28	**132**	**38**	138	2899	0.98	487	159	124	8245	0.29	4820	66	1
29	**61**	**23**	37	690	1	282	77	61	4115	0.27	305	67.4	3
30	**565**	**70**	920	5754	0.77	1435	488	372	15776	0.24	2165	42	1
Mean	125	28	110	1923	0.77	357	111	113	4724	0.34	2890	69	3

Appendix B: JSprit's Optimization Algorithm

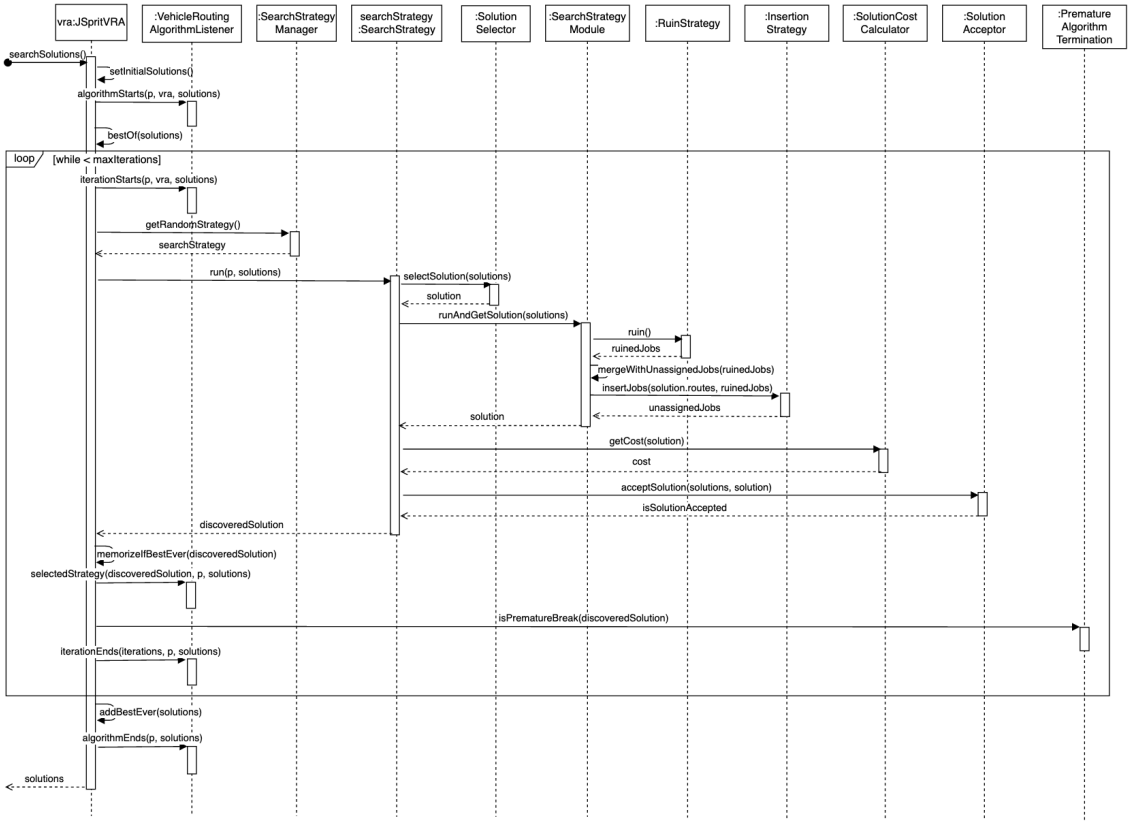


Figure 1: Search process of the *JSprit* solver. When the searchSolutions() starts, it's assumed that all the required configurations and data was previously provided to the object *vra*.

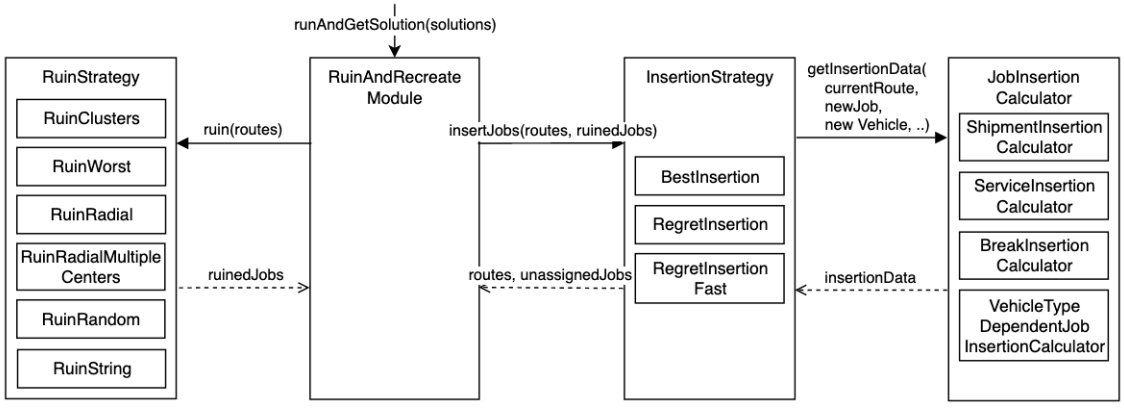
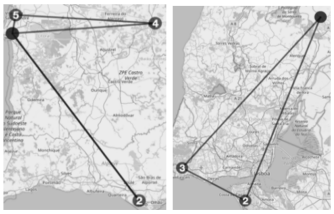
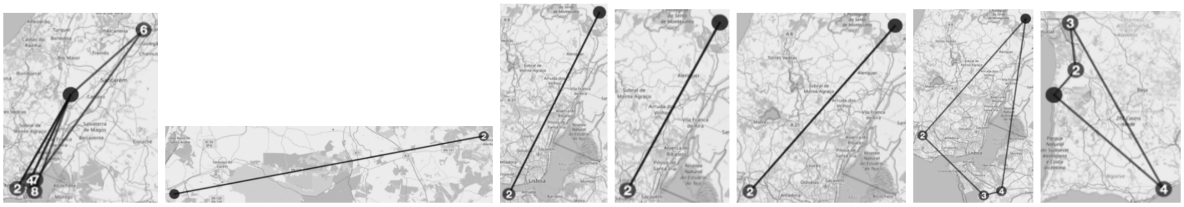


Figure 2: Main components and interactions in a ruin-and-recreate iteration of *JSprit*. The steps are inspired by the ruin-and-recreate principle from Gerhard Schrimpf 2000.

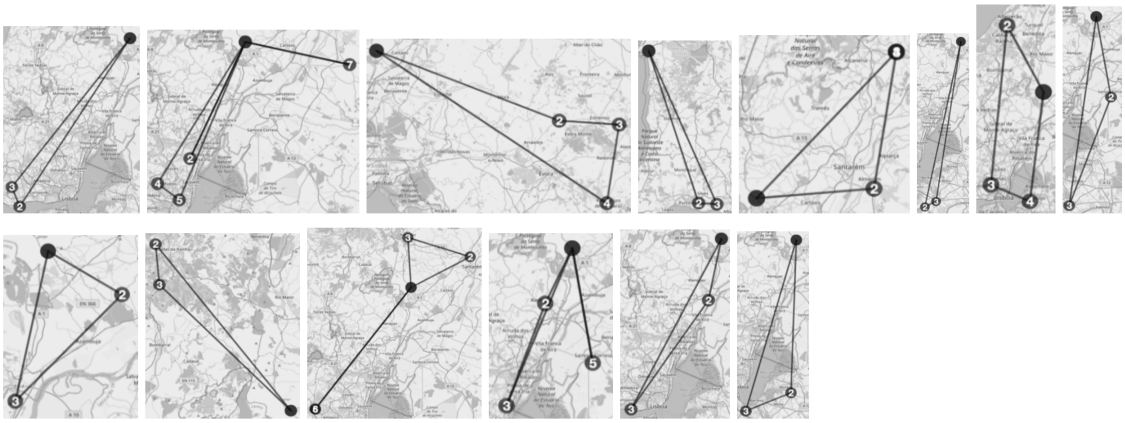
Appendix C: Routes Classification



(a) Routes ranked as Bad



(b) Routes ranked as Acceptable



(c) Routes ranked as Good

Figure 3: Classification of the routes for the final solution. Total of 23 routes, 2 (9%) routes ranked as bad, 7 (30%) as acceptable and 14 (61%) good. The numbers represent the order of the deliveries.

Appendix D: Used Software Libraries

- **Report Generation:** To notify the progress of the fuel model to the client reports were generated with the most recent versions of the Python interpreter and the following libraries between October and December 2020.
  - **Matplotlib:** Creates static and animated data visualizations. Used to generate a Gantt chart (textitgithub.com/matplotlib/matplotlib);
  - **Requests:** Allows to send HTTP requests. Used to obtain distances and durations (*github.com/psf/requests*);
  - **FPDF:** PDF document generator ported from PHP FPDF. Used to generated the report (*github.com/reingart/pyfpdf*);
  - **Folium:** Data visualizer in a leaflet map. The leaflet is a Javascript library to build interactive maps. Used to generate the maps (*github.com/python-visualization/folium*);
  - **PIL:** Imaging library that adds image processing capabilities to the Python interpreter. Used to convert the maps in HTML format generated by Folium to png format.
- **API Gateway:** A web application developed using the Django Framework, the following libraries were used.
  - **Django v3.1.3:** Python web framework following a model-template-views architectural pattern. (*github.com/django/django*);
  - **Django-rest-framework v3.1.3:** Enables Django to work as a REST framework (*github.com/encode/django-rest-framework*);
  - **Pika v1.1.0:** A python implementation of the Advanced Message Queueing Protocol (AMQP) (*github.com/encode/django-rest-framework*);
  - **Celery v4.1.1:** Asynchronous job queues implementation to process jobs in dedicated threads or processes. Celery requires a message broker to send and receive messages, therefore RabbitMQ is used (*github.com/celery/celery*);
  - **RedisLock v3.6.0:** Distributed locking using Redis, a in-memory database. Thus, a lock can be shared across different processes (*github.com/ionelmc/python-redis-lock*);
  - **Raven v6.10.0:** A python client for Sentry, a cloud based error monitoring (*sentry.io*) that *Sentilant* uses in the development of its software;
  - **Psycopg2 v2.8.6:** PostgreSQL database adapter for python to allow Django access the database (*github.com/psycopg/psycopg2*);
  - **Watchdog v0.10.3:** Listen system events and allows to auto-restart services. Used in development to automatically apply code changes to celery workers; (*github.com/gorakhargosh/watchdog*)
  - **Django Swagger UI v0.1.11:** Swagger UI for Django, the API was documented in a *OpenAPI Specification* file, this library reads the file and generates an HTML single-page documentation (*github.com/assem-ch/django-swagger-ui*).
- **Field-Service Planning System:** Part of the planning system is containerized using *Docker* (*docker.com*) and its composed by the following containers.

  - **RabbitMQ Server v3.8.9 with Management Plugin:** Message broker configured with the AMQP to establish communication between the *API Gateway*, the solvers and the *Celery* workers; The management plugin allows to monitor the server with an User Interface (UI) interface (*hub.docker.com/_/rabbitmq*);
  - **PostgreSQL v13:** Relational database (*https://hub.docker.com/_/postgres*);
  - **Redis v6.0.9:** In-memory NoSQL key-value store that can be used as database, cache and message broker. In this work Redis is used together with the *Redis-Lock* library to prevent simultaneous write/reads performed by the different *Celery* workers (*hub.docker.com/_/redis*);
  - **Web App:** Python web framework following a model-template-views architectural pattern. Django is transformed in a REST API with Django Rest Framework, in this work v3.12.2 is used to implement the API Gateway;
  - **NginX v1.19:** HTTP Web Server to deploy the Django application (*hub.docker.com/_/nginx*);
  - **Worker:** Celery worker dedicated to consume the work plans obtained by the solver, save in the database or return to the client; A lock is created when a problem is submitted to be solver, the lock key is obtained by the worker to prevent two or more workers to read/write simultaneously;
  - **Redbeat Server:** Celery beat scheduler that stores the scheduled jobs and runtime metadata in Redis, so the data isn't tied to a single drive or machine. A modified version by *Sentilant* is used (*github.com/sibson/redbeat*).

- **Valhalla v3.0.9:** Routing engine to compute duration and distance between locations, similar to OSRM it uses geodata in the OpenSourceMap (OSM) format. Configured with the map of Europe from *feofabrik.de*, a 22.5 Gb OSM file updated at December 12th 2020. (*github.com/valhalla/valhalla*); Valhalla allows to exclude locations, meaning that no route will pass through that location e.g., Ponte 25 de Abril;

- **JSprit v1.7.1 (2017-05-11):** Version of the VRP solver forked by *Sentilant*;

- **Jupyter Notebook:** Python web application to build code live notebooks, used for an experiment to compare *OSRM* and *Valhalla* to verify if Ponte 25 de Abril was avoided and data analysis.