FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Disruption Management of ASAE's Inspection Routes

Miguel Milheiro Pinto Ferreira

WORKING VERSION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Luís Paulo Reis

Second Supervisor: Henrique Lopes Cardoso

October 31, 2021

Disruption Management of ASAE's Inspection Routes

Miguel Milheiro Pinto Ferreira

Mestrado Integrado em Engenharia Informática e Computação

October 31, 2021

Abstract

The emergence of technologies capable of producing real-time data opened new horizons to planning and optimisation of vehicle routes. Dynamic vehicle routing problems (DVRPs) make use of real-time information to calculate the most optimised set of routes at a certain moment. DVRP is a challenging problem because its scope is real-time, meaning that decisions sometimes must be made in short time windows, preventing the use of complex algorithms that require long computational times. The typical approach to this problem is to initially calculate the routes for the whole fleet and dynamically revise them in real-time once a disruption occurs.

In this dissertation, the Autoridade de Segurança Alimentar e Económica (ASAE) operational inspections will be explored and analysed as a case study of the DVRPs. ASAE is a Portuguese administrative authority specialised in food security and economic surveillance and regulates millions of economic entities in the Portuguese territory. ASAE's inspections are usually done by brigades using vehicles to visit economic operators. Project IA.SAE, an exploratory project with scientific purposes, gave rise to a route optimisation module capable of defining and assigning routes to inspect economic operators, seeking to maximise a utility function. This dissertation arises in the context of project CIGESCOP, which tackles the same problem and aims to build an application to be launched in production. Optimisation algorithms calculate inspection routes for each brigade, with information regarding specific map paths and inspection schedules. However, the approach used in project IA.SAE does not consider the dynamic properties of real-life scenarios, which means that the precalculated operation plan is not reviewed in real-time. This work aims to study the dynamic properties of ASAE's operational environment and proposes a solution to efficiently review the precalculated inspection routes and apply the required changes in an appropriate time frame.

This work will model the problem as a DVRP and compare the performance of four metaheuristics in a series of metrics: Hill-Climbing, Simulated Annealing, Tabu Search and Large neighbourhood Search. This work proposes a weighted utility function based on three aspects: the sum utility of the customers visited, the similarity between the initial and new solutions, and the time when the brigades reach the final depot at the end of the workday. The approach implemented in this work includes a Disruption Generator module, developed to simulate and generate disruptions on the inspection routes randomly and with defined intensities. The disruptions taken into account were: travel and inspection times, vehicle and inspection breakdowns, utility changes, and emergency inspections.

All the algorithms solved most of the several problem instances tested, providing reasonable solutions. The Hill-Climbing algorithm had the fastest convergence, while the Simulated Annealing took the most time to solve the test instances. The Large Neighborhood Search was revealed to be the method with higher solution quality, while the Hill-Climbing provided solutions of lower

utility.

Keywords: Vehicle routing, Disruption Management, Real-Time Scheduling, Routes Rescheduling, Hill-Climbing, Simulated Annealing, Tabu Search, Large Neighbourhood Search

Resumo

O aparecimento de tecnologias capazes de produzir dados em tempo real abriu novos horizontes para o planeamento e otimização de rotas para veículos. Dynamic Vehicle Routing Problems (DVRPs) dão uso a informação disponível em tempo real calculando o conjunto mais otimizado de rotas num determinado instante. Os problemas DVRP são mais desafiadores na medida em que são problemas passíveis de resolução em tempo real. As decisões e soluções têm de ser apresentadas em intervalos de tempo curtos, impossibilitando o uso de abordagem mais complexas, como algoritmos que requeiram elevados tempos computacionais. A abordagem típica para este problema é calcular à priori um conjunto de rotas para a frota de veículos e, em tempo real, rever o plano de operações definido, modificando-o sempre que ocorra um evento disruptivo.

Nesta dissertação, o cenário de inspeções operacionais da Autoridade de Segurança Alimentar e Económica (ASAE) servirá como caso de estudo para um DVRP. A ASAE é uma autoridade portuguesa especializada em segurança alimentar e auditoria económica, sendo atualmente responsável por regular milhares de agentes económicos em solo português. As inspeções realizadas pela ASAE são efetuadas por brigadas que se deslocam em veículos e têm como objetivo fiscalizar operadores económicos. IA.SAE foi um projeto exploratório com propósitos científicos, que culminou no desenvolvimento de um módulo de otimisação de rotas capaz de definir e atribuir um conjunto de rotas de inspeção a um conjunto de agentes económicos, maximizando uma função de utilidade. Esta dissertação surge no contexto do projeto CIGESCOP, que aborda o mesmo problema e tem como objetivo o desenvolvimento de uma aplicação para ser lançada em produção. No âmbito do projeto IA.SAE foram usados algoritmos de otimização para calcular as rotas de inspeção de cada brigada. O sistema implementado utiliza a solução produzida por algoritmos de otimização para representar as rotas calculadas num mapa e criar tabelas com o horário de inspeções. No entanto, a abordagem usada não considera as propriedades dinâmicas de um ambiente real e o plano operacional não é revisto em tempo real. Este trabalho de dissertação tem como objetivo estudar as propriedades dinâmicas do ambiente em que a ASAE opera e propor uma abordagem que permita, de forma otimizada, rever o plano de operações previamente calculado e fazer modificações necessárias em tempo real. O cálculo do novo plano deve ser feito num período temporal aceitável, o que implica o uso de métodos para aproximação da solução ótima.

Esta dissertação irá modelar o problema como um DVRP e tem como intuito comparar a performance de diversos métodos meta-heurísticos: Hill-Climbing, Simulated Annealing, Tabu Search, e Large Neighborhood Search. Este trabalho propõe uma função de utilidade flexível e resultante de três diferentes domínios: soma de utilidade resultante dos clientes visitados, similaridade da nova solução com a solução inicial, e tempo de chegada de cada brigada no final do dia de trabalho. A abordagem implementada nesta dissertação inclui um módulo de geração de disrupções, desenvolvido com o intuito de simular e gerar disrupções nas rotas de inspeção de forma aleatória e com intensidades definidas. As disrupções consideradas neste trabalho são: tempo de viagem e inspeção, colapso de veículos e inspeções, mudanças nas utilidades, e inspeções de emergência.

Os algoritmos desenvolvidos resolveram em globalidade as instâncias do problema usadas como teste, apresentando soluções razoavelmente boas. O algoritmo Hill-Climbing apresentou uma convergência mais rápida, em contraste com o Simulated Annealing que utilizou um intervalo de tempo maior para encontrar a melhor solução. O método Large Neighborhood Search foi aquele que apresentou melhores soluções, enquanto o Hill-Climbing obteve soluções de menor utilidade.

Keywords: Roteamento de veículos, Gestão de rupturas, Agendamento em tempo real, Reprogramação de rotas, Hill-Climbing, Simulated Annealing, Tabu Search, Large Neighbourhood Search

Agradecimentos

Este trabalho teve como orientador o Professor Luís Paulo Reis, que me permitiu trabalhar neste tema e desde início me fez sentir integrado no projeto. Apesar das dificuldades acrescidas devido ao contexto pandémico, o Professor demonstrou sempre um auxílio bastante competente que ajudou em muito o curso desta dissertação. Agradeço também ao coorientador Henrique Lopes Cardoso, por toda a ajuda que prestou durante o curso deste trabalho.
À Faculdade de Engenharia da Universidade do Porto e a todo o Departamento de Engenharia Informática pela casa que foram durante estes cinco anos da minha vida académica. Também por todas as condições e ajudas para a minha formação académica. Um agradecimento também a todos os Professores que me trouxeram bastantes conhecimentos nesta área, bem como conhecimentos a nível pessoal e sobre o mundo do trabalho.
À ASAE e em especial aos seus membros que através do seu feedback e sugestões permitiram a realização de um trabalho que espelha o seu modo de operação.
Um agradecimento muito especial ao meu Pai, uma pessoa excecional que por infelicidade partiu, mas sem dúvida deixou a sua marca no mundo e contribuiu para a pessoa que sou hoje. A ele um muito obrigado por todos os momentos que passamos juntos.
Um agradecimento também a toda a minha família, em especial á minha mãe, irmãos e avós, que sempre demonstraram o seu apoio, incentivo e paciência durante todo o meu percurso. Um agradecimento também a todas a suas influências, que me tornaram a pessoa que sou hoje. Sem eles nada seria possível.
Por fim um agradecimento aos meus amigos, uns mais próximos que outros, mas todos com uma grande influencia na pessoa que sou hoje. Um obrigado também por todos os bons momentos que passámos juntos, os quais guardarei para sempre na minha memória. Agradeço também áqueles que me acompanharam no meu percurso académico e foram uma ajuda essencial para a conclusão desta fase da minha vida.

Miguel Ferreira

"If I had asked people what they wanted,
they would have said faster horses."

Henry Ford

Contents

List of Figures

List of Tables

Abbreviations

ALNS	The Adaptive large neighbourhood search
ACS	Ant Colony Systems
ADP	Approximate Dynamic Programming
API	Application Programming Interface
CIGESCOP	Centro Inteligente de Gestão e Controlo Operacional da ASAE
CVRP	Capacitated Vehicle Routing Problem
DDVRP	Dynamic and deterministic Vehicle Routing Problem
DSVRP	Dynamic and stochastic Vehicle Routing Problem
DVRP	Dynamic Vehicle Routing Problem
GPU	Graphics Processing Unit
HC	Hill-Climbing algorithm
IA.SAE	Inteligência Artificial na Segurança Alimentar e Económica
LNS	Large Neighborhood Search
LIACC	Intelligence and Computer Science Laboratory
MDDVRPTW	Multi-depot Dynamic Vehicle Routing Problem with Time Windows
MDVRP	Multi-depot Vehicle Routing Problem
NP-Hard	Non-deterministic polynomial-time hardness
RTS	Reactive Tabu search algorithm
SSD	Solid State Drive
TS	Tabu search
TOP	Team Orienteering Problem
TSP	Traveling Salesman Problem
VRP	Vehicle Routing Problem
VRPPD	Vehicle Routing Problem with Pickups and Deliveries
VRPTW	Vehicle Routing Problem with Time Windows
VLSN	Very Large Scale Neighborhood search

Chapter 1

Introduction

In this chapter, an introductory overview of the work is performed analysing several vital aspects: the context, motivations, related project, and objectives. The document structure is also discussed.

1.1 Context

This work arises from a collaboration between the *Artificial Intelligence and Computer Science Laboratory (LIACC)* and *Autoridade de Segurança Alimentar e Económica (ASAE)*. It is related to the route optimisation module in the project *Centro Inteligente de Gestão e Controlo Operacional da ASAE (CIGESCOP)*. Project CIGESCOP can be seen as the second iteration of a preliminary project by the name of Inteligência Artificial na Segurança Alimentar e Económica (IA.SAE). This project had the primary goal of improving ASAE's operations by allying modern machine learning techniques and artificial intelligent in an information system.

ASAE inspection scenario will be used as a case study for this work. ASAE is a Portuguese authority specialised in the areas of food safety and financial supervision. ASAE is responsible for inspecting thousands of economic entities of the Portuguese territory and report or take actions about non-compliance with the law. Their activities are of extreme importance as they ensure competition fairness among all the economic entities and improve food safety. ASAE inspects the economic operators, both proactively (planned inspections) and reactively (unplanned inspections). Agents are periodically inspected for non-compliance with the law or in the case of investigations and inquiries, respectively. To serve this purpose, ASAE has a substantial amount of vehicles and brigades to operate.

1.2 Projects IA.SAE / CIGESCOP

Project *IA.SAE* is one of four projects financed by the Portuguese government in the context of program *Incode 2030*.[1] This program has an international scope and aims to improve Portuguese compatibility capacities with digital competences. *IA.SAE* aims to promote food safety, consumer protection and fair competition between the economic operators in the Portuguese territory. To accomplish this, it relied on the development of risk assessment models combined with appropriate machine learning and artificial intelligence techniques to select the most promising set of economic operators to inspect. This selection is based on a complex utility function that gathers and weights the ASAE databases' information, for example, past complaints. This information was gathered and combined using appropriate machine learning methodologies (both in data and text mining), resulting in valuable knowledge extraction. The number of complaints targeted at a particular economic operator, past crimes, and macro-risks forms examples of crucial factors taken into account. Routes are then assigned in an optimised way to the respective vehicles and brigades, maximising the utility function. *IA.SAE* is segmented in five distinct modules: a system to generate and optimise the risk matrices, a system to analyse complaints and reports, a system to help in the supervision of prices namely on promotions, a system to make an intelligent selection of the economic operators to inspect and a system do visualise several performance metrics and geoinformation.

IA.SAE was a exploratory project with scientific purposes, while *CIGESCOP* is a project that aims to build an application to be launched in production. Project *CIGESCOP* has the main objective of optimising inspection processes by endowing brigades with tablets that generate information and communicate with a centralized intelligent control system. Most of the preliminary work will be adapted and improved in this new version of the system. Concerning this dissertation, the route optimisation module is of significant importance. This module calculates the set of optimised routes to visit several economic operators, maximising a utility function. The routes for an operation day are calculated, and a plan is produced. The respective brigades then accomplish this plan. The current approach does not take into account dynamic factors that are present in the real world scenarios and might cause disruptions. As any system that operates in a real-life scenario, ASAE brigades are subject to factors that might delay or preclude the precalculated operation plan. These factors will be named disruptions, and some examples are delays on inspections, vehicle breakdown, close roads, and traffic intensity.

1.3 Motivation

The contributions and approach proposed by this work can be used in a panoply of scenarios and institutions that operate in a similar way, improving the already implemented approaches on each case. Addressing dynamic factors in a particular environment will further optimise the vehicle routes, as disruptions will no longer threaten the static route plan. A static route plan can become

[1] Project Incode2030 - Metas, available at https://www.incode2030.gov.pt/metas, accessed on 2021-01-02

unfeasible if specific disruptions are not addressed. For example, a vehicle breakdown will prevent unserved customers from remaining unvisited if the plan is not revised.

The current pandemic scenario also plays as motivation, since the market share for deliveries has considerably increased, and companies are demanding more intelligent vehicle routing systems. A system capable of mutating the vehicle routes and create dynamic plans, compared to the classic static plans will mean further optimisation in the vehicles' routes, entailing all the economic and environmental benefits.

Addressing dynamic elements will imply that real-time information must be collected during the plan's execution and used by the routing system to tackle the disruptions. Real-time information systems are evolving, becoming considerably affordable and more portable.

One motivation behind this work, is to use ASAE as a case study, optimising and improving the current ASAE's inspection process. Optimising an organisation's processes with such significant importance to Portuguese citizens as ASAE will have many benefits to society. The economic operators with a higher probability of non-compliance with the law will be assigned with a higher weight on the utility function, increasing the inspection priority. Brigades will readily identify and sanction infringements of the law.

The project will increase food security directly and indirectly. Legal sanctions taken against specific economic operators will rectify wrong policies, and an "intelligent system" will discourage economic operators from facilitating in certain aspects, respectively. Also, as the routes will be optimised, more economic operators can be inspected on a day's work. The vehicle's ecologic footstep will also be reduced.

Several entities that operate in a similar way with ASAE might also benefit from this work's contributions. The approach used to tackle the dynamic factors during the route execution can be easily adapted to similar scenarios.

1.4 Objectives

This dissertation work proposes an approach that, by using real-time information, revises vehicle route plans and makes the necessary changes to address dynamic factors. This work proposes a set of algorithms capable of calculating new route plans, modifying previous calculated plans once a disruptive event occurs, aiming to maintain the optimal plan. An objective function is proposed as the optimisation criterion, calculating each customer's utility upon visit. The utility function is a complex function that depends on three different weighted domains to calculate the utility of a particular solution.

The objectives for this thesis are also in the scope of ASAE, as it is used as a case study. The routing module developed in the context of IA.SAE calculates the inspection routes before their execution and does not implement the routes' revision in real-time. Some delays and disruptions will invalidate the precalculated routing plan as the current system cannot address these issues. The existence of several systems for collecting data in real-time, such as the geographic coordinates of each operating vehicle, facilitates route revision. Using the information provided by such methods,

this work implements an approach that can revise and recalculate the routes in real-time once disruptions occur. Dynamically optimising the routes in real-time leads to further optimisation as the system takes advantage of opportunities to further optimise the routing. For example, if an economic agent is closed, a static approach cannot immediately reroute the brigade to another good inspection location.

This dissertation compares the performance of four algorithms proposed to solve this optimization problem. Several system runs with specific test conditions will allow conclusions about the efficiency of this approach in addressing all kinds of disruptions considered in this work.

1.5 Document Structure

This dissertation document is divided into eight additional chapters chapters. Chapter 1 gives a brief introduction to this work, providing motivations and contextualising the work. It also gives a brief insight into the case study that will be used during the development of this work.

The following three chapters pertain to the review on the state-of-the-art. Chapter 2 reviews the state-of-the-art on Vehicle Routing Problems (VRPs), focusing on its dynamic variant, the Dynamic Vehicle Routing Problem (DVRP). Chapter 3 reviews the literature on the four optimization algorithms later implemented on the proposed solution. This chapter gives a brief insight into the fundamental concepts behind each algorithm and explores various improvements made to enhance the performance of their basic implementations.

Chapter 4 explores the case study that will be used in this dissertation. It features the implementation previously done in the context of project IA.SAE, explaining the problem formulated as a VRP. This chapter also reviews the use of artificial intelligence techniques in the food safety context, focusing on the use of Big Data.

Chapter 5 describes the problem related to this dissertation work and proposes a solution approach based on the previous chapters. The next two chapters describe the implemented approach. Chapter 6 concerns the Disruption generator module. It generates random disruptions to the inspection routes that the algorithms will later solve, outputting a new optimised plan. This chapter will go into detail on the disruption types considered in this dissertation and their respective implementation.

Chapter 7 is one regarding the implementation of the main part of the system. It describes the routing application used to calculate the travel times between two economic operators and the framework used for the map visualization. It then describes the web application developed and how it shows the information coming from the other modules. This chapter also gives an insight into the data structures used for representing several problem abstractions. The economic operators' schedules are described, and the method used for their generation is explained. This chapter also describes the utility function used in this approach and the allowance of unfeasible solutions with associated penalties. Finally, this chapter presents the solution generation methods used by all the algorithms to progress in the search.

Chapter 8 regards the results and analysis over a set of experiments. The system ran in several test configurations, and a series of metrics were registered and analysed. Additionally, a comparison between the several algorithms and disruptions types is performed based on the results. Finally, in the last chapter, the conclusions and results of this work are briefly presented. Several other considerations are explored, such as the limitations and future development perspectives regarding this work.

Chapter 2

Dynamic Vehicle Routing Problems (DVRP)

This chapter will review the state-of-the-art on Dynamic Vehicle Routing Problems. Initially, a brief definition of this problem's static version, Vehicle Routing Problems (VRPs) will be discussed, and the differences between the two variants (static and dynamic). The chapter will also review a taxonomy on DVRPs and different formulations and variations of this problem. Finally, examples of solution methods and approaches will be reviewed.

2.1 Introduction

During the last decades, urban transportation experienced a rapid and significant evolution supported by several vital technologies' emergence and development. On the other hand, the computing hardware's growth, supported by an increase in the number of transistors in dense circuit boards. This resulted in approximately double the computational power every two years since 1975 (Moore's Law[1]). Computational power also benefits from new computation paradigms such as parallel computing. Specific portions of the code are executed in the Graphics Processing Unit (GPU), massively boosting performance.[2] Lastly, the Disk drive's capacities increased, and their performance massively increased with the rise of technologies like Solid State Drive (SSD). These technologies and processes arise the opportunity to improve vehicle performance, mainly by optimising the vehicle's routes. Several benefits can be accounted for: improved safety, less traffic congestion, monetary savings, and environmental impacts [17]. The typical approach relies on centralised control, having a control infrastructure communicating, and gathering vehicle data. The control infrastructure combines vehicle data, leading to more efficient and intelligent route-optimisations. Concerning dynamic environment factors, further optimisations can be achieved by the use of algorithms that benefit from the information collected in real-time. The evolution

[1] Moore's law - Computer science, available at https://www.britannica.com/technology/Moores-law, accessed on 2021-01-29

[2] What is Parallel Computing?, available at https://www.omnisci.com/technical-glossary/parallel-computing, accessed on 2021-01-29

of communication mediums eases knowledge transference between the vehicle feet and the centralised control.

2.2 Vehicle Routing Problems (VRP)

With the optimisation of the vehicle routes, the vehicle fleet capacities are being explored to the theoretical maximum, improving the balance between a particular utility function and the operating costs. Most common approaches so far pre-calculate the operations plan, taking into account only the information available at the time. Routes are generated before the vehicle fleet executes the plan, meaning state-of-the-art complex algorithms can be used to get a better solution. Computational times are usually negligible in this case. The Vehicle Routing Problem (VRP) consists of determining the set of routes to be traversed by a fleet of vehicles to serve a set of costumers or to visit a set of locations [15]. It was first introduced by Dantzig and Ramser (1959) [13]. Since then, multiple modulations of the problem and algorithms to approximate solutions were described in the literature. The most straightforward and famous vehicle routing problem is the Traveling Salesman Problem (TSP): Having a set of cities to visit, calculate the shortest path, starting from an initial city, that visits each city exactly once and then returns to the starting city [22]. Several variants of the VRP were proposed in the literature, such as Capacitated VRP (CVRP), VRP with Time Windows (VRPTW), and Multidepot VRP (MDVRP).

VRP and all its variants are NP-Hard problems, as they cannot be solved in polynomial time [15]. Heuristics and other algorithms are used to calculate the approximate solution when this is sufficient. To the literature, the term VRP usually appears associated with static environments. Static VRPs approximate the real-life scenario as they reduce the real problem to a static setting. The obtained solution does not consider dynamic factors that can influence the pre-calculated operations plan during its execution. The set of routes outputted when solving the VRP might be impossible to execute as some unforeseen events happen. Also, there might be a window to optimise the pre-calculated operations plan further. The literature describes this uncertain factors as disruptions: "the action of preventing something, especially a system, process, or event, from continuing as usual or as expected".[3] Several types of disruptions were addressed in the literature: vehicle breakdown disrupted links in the road network, variations in the supply of goods, variations of customer demand [15], and service and travel times.

2.3 Dynamic Vehicle Routing Problems (DVRP)

Research on the field of vehicle routing has increased massively, with enterprises aiming to lower their costs and increase their profits. This area attracted many researchers, mainly on the subject of dynamic routing, especially in the last three decades [43]. Pillac, on his survey, catalogued 154 references on the topic, which confirms the trend. [38] (Figure 2.1).

[3] Cambridge Dictionary - disruption, available at https://dictionary.cambridge.org/dictionary/english/disruption, accessed on 2021-01-29

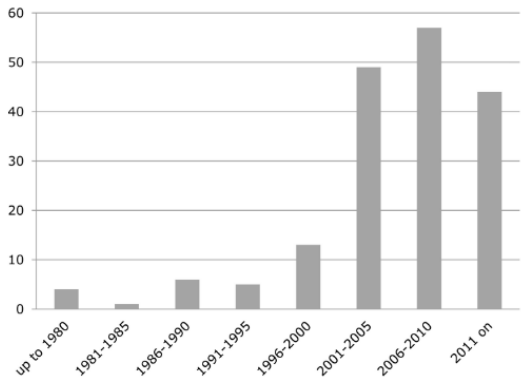


Figure 2.1: Evolution of published papers regarding DVRPs. [43]

The hardware evolution plays an essential role in this trend, as previously referred. On the other hand, the appearance of devices and systems that can gather and transmit vehicle data in real-time foster new implementations of DVRPs. Dynamic vehicle routing relies on the system's capability to grasp and perceive the environment's dynamic factor. The emergence of the Internet provides a basis for easier information dissemination and provides a valuable framework for connecting all the nodes involved in vehicle routing (vehicles, control system, external systems). The Internet also allowed access to Application Programming Interfaces (API), that provide useful information to vehicle routing. Information such as traffic intensity, meteorology events that affect vehicles, demand and production forecasts and factors that influence the vehicle operating costs. Systems like Global Positioning System (GPS) also played a crucial role as they provide every vehicle's exact position in the fleet. Lastly, the dissemination of mobile devices like smartphones and other devices capable of gathering essential data in real-time [38]. These technologies foster communication among connected vehicles operating in the field, also allowing them to communicate with a control facility. This subject is connected to the field of Big Data, which has drawn more attention from operations researchers in recent times. Companies and enterprises gather massive amounts of data as the numerous connected devices are more common and increasingly easier to afford [43]. These devices can gather large data streams without human intervention, and are easier to correlate and interpret, as the computational power still grows exponentially. All these data and systems can be used and combined in real-time to gather and create useful information that will play a crucial role in optimising vehicle routes. They enhance and explain the decisions taken in Dynamic Routing. Dynamic Vehicle Routing Problems (DVRP) were first described by Wilson and Colvin (1997) [55]. They studied the dial-a-ride problem (DARP), where client requests appear dynamically during execution time. They used an insertion heuristic approach to obtain an approximate solution with low computational effort [43]. DVRPs presuppose that the routes can be continuously updated, adapting to the uncertain environment circumstances, maintaining their optimality as far as possible.

In the literature, and as addressed below (Section 2.5) on the taxonomy proposed by Psaraftis [43],

DVRPs are divided into two clusters: the Dynamic and deterministic VRP and the Dynamic and stochastic VRP. Although both variants can be considered dynamic, they differ in the presence of stochastic information about dynamic events. In Dynamic and stochastic VRPs, this information is available beforehand and may be useful to plan future decisions.

2.3.1 Dynamic and Deterministic VRP

This set of problems involve dynamic scenarios where all the inputs are known with certainty; no stochastic inputs. The inputs in this kind of problems are still dynamic, as they are unknown in advance. The big difference from these types of problems compared to dynamic and stochastic VRPs is that no stochastic information is known about a future event [43]. There are no probabilities or probability distributions capable of predicting or weigh any upcoming event. For example, there is no available data in a travelling repairman's context that may infer how much time the repairman will take on each customer service. The totality of information about an event is only available upon it happens.

2.3.2 Dynamic and Stochastic VRP

Dynamic and stochastic VRP (DSVRP) are a particularly interesting problem variant as they handle real-work scenarios more accurately [44]. Apart from handling the dynamic information available over time, they consider stochastic knowledge to make decisions. Ritzinger et al. [44] points the field of anticipatory optimisation as being related to dynamic decision making: a single decision in a given a sequence of interdependent decisions impacts the entire decision process. Ritzinger also refers the reader to the classification in two groups regarding different degrees of anticipation concerning DSVRP: prepossessing decisions or online decisions.

Prepossessing decisions means solutions are computed before the route execution, while online decision presupposes that solutions are computed whenever a dynamic event occurs. In the first group, all the possible problem states need to be calculated based on future dynamic events that might happen and the stochastic information about them. When executing the operations plan, the states and policies previously defined work like "rules" that are used to tackle the dynamic events. An example of a "rule" is to always assign the routes of a broke-down vehicle to the closest operating vehicle in the road network. A second variant of prepossessing decisions works in a similar way, but instead of "rules", it attributes a value to states and their corresponding decisions. During the execution phase, the system uses the precomputed values to make decisions. Prepossessing decisions focus the intensive computation tasks before the route execution.

When using online decisions, a significant part of the computation is done when a dynamic event occurs. [44] The solution is calculated during the execution phase of a precalculated static plan. Whenever a dynamic event arises, the system uses the available stochastic information to make decisions. Ritzinger refers to this process as a "rolling horizon" or "look-ahead". The dynamic solutions are calculated by either re-optimising the plan or addressing specific situations, outputting one decision upon a particular dynamic event. The system can also opt for a single

greedy decision upon dynamic events, allowing for fast-responsiveness at lower computational efforts [44]. The greedy interim solution is then further optimised in the background and is later provided to the system.

2.3.3 Differences with Static Routing

DVRPs have more degrees of freedom than VRP, as they involve new factors that increase route decisions' complexity. The worthiness of a route plan is also harder to judge [38]. As a consequence of dynamic factors, the system can deny requests either because they are unfeasible or the penalisation of accomplishing them is too high. This process was referred to as *service guarantee* [24]. Dynamic routing usually has discrepancies and addictions to the objective functions. Although maintaining the fundamental static routing objective of minimising the route costs, dynamic routing extends it to reducing the costs of recalculating a route. The cost of deviating from the original plan can be hard to quantify. Deviation costs can be divided into two main clusters: costs to the customers and costs to the drivers [15].

Customers plan their schedules to receive certain services or commodities at a particular time. Deviations can cause them losses and increase their degree of dissatisfaction as the pre-determined delivery time changes. For the drivers, deviations from the original plan can mean special payments or the use of overtime. Drivers may have personal costs, that arise because they might be unfamiliar with a customer, process, or road network. This inexperience can lead to an increase in service time when compared to the original plan [15].

The complexity of DVRP also derives from unforeseen factors not being known before their manifest. Routes must be calculated close to real-time, while the vehicle fleet is operating, leaving a small time window to perform the needed optimisations.

With the evolution of the Machine Learning domains and using the prediction capabilities of nowadays state-of-the-art machine learning models, it is possible to produce predictions close enough to reality. Systems can forecast disruptions and calculate the best set of routes according to their forecasts, simulating a dynamic world. The routes would be calculated beforehand, which allows them to use higher computational time algorithms that usually output better solutions. Although this seems a feasible approach, Taniguchi presents convincing results that refute it on a study of a VRP with time window (VRPTW) [53]. In this paper, two approaches are compared: a forecasted VRPTW (VRPTW-F) and the implementation of a Dynamic VRP with time windows (VRPTW-D). On their results, a 3.7% improvement on the costs is reported when using the dynamic problem formulation (VRPTW-D) compared with the VRPTW-F. They also refer that by incorporating real-time information of the fleet travel times, they improved the customer service, as the arrival times to the customers are better enclosure in the defined time-windows. This contributes to reducing the delay penalties. Finally, the VRPTW-D is reported as more effective in decreasing the vehicles' running time when compared to the forecasted version. VRPTW-Ds can, therefore, contribute to less road congestion and reduce the vehicles' operating costs.

Dynamically calculating routes is not a trivial problem, increasing complexity with the number of vehicles on the fleet, the number of customers to serve, and other problem-related restrictions.

As vehicles are operating, routes need to be modified and recalculated in close to real-time once a disruption appears. The time available can be too short for the use of complex state-of-the-art algorithms. One approach to address this problem is to calculate the provisional operations plan with the available information at a specific time. This plan is then revised once disruption occurs. The concept of Disruption Management is to revise an operation plan in real-time once disruption occurs dynamically [15]. It has major importance when the operations plan has been calculated in advance, or its execution is vulnerable to significant disruptions.

2.4 Disruption Management

A formal definition of disruption management was present by Yu and QI (2004):

> ”At the beginning of a business cycle, an optimal or near-optimal operational plan is obtained by using certain optimisation models and solution schemes. When such an operational plan is executed, disruptions may occur from time to time caused by internal and external uncertain factors. As a result, the original operational plan may not remain optimal, or even feasible. Consequently, we need to dynamically revise the original plan and obtain a new one that reflects the constraints and objectives of the evolved environment while minimising the negative impact of the disruption. ” [58]

In his survey, Eglese clarifies four factors directly involved in disruption management [15].

The time available for the replanning and re-optimisation is usually limited. A particular algorithm uses a portion of this time to output the revised plan; time limitations advise against algorithms with high computational time. Authors also highlight the fact that the revised plan needs to be communicated to the vehicle fleet. The time employed in the communication of a revised plan should also be taken into account in the time available for replanning [15].

In disruption management scenarios, there is always access to the original pre-calculated operations plan. This plan is useful and serves as a starting point for calculating the disrupted plan, as there is no need to recalculate everything from the start. Using a previously calculated operations plan is a fundamental argument to stint the short time available for replanning [15]. Overall, the optimality of the solution benefits from this approach. At an early step, a solution can be generated based on a static implementation of the VRP. As this solution is generated previously to the operations plan’s execution, more expensive and complex algorithms might be used, generally outputting a better solution. Once disruption occurs, the plan is revised and modified, but the initially calculated plan serves as a starting point. For instance, an algorithm can calculate all the vehicle routes for the whole workday during one night. This plan is then revised in real-time and altered according to the occurrence of disruptions.

As the initially calculated plan will be modified and revised, the costs entailed by deviations should also be taken into account. Therefore, Eglese refers that disruption management is often a multi-objective problem, as the costs of modifying the plan should be added to the costs already present on the original plan (operation and other costs) [15].

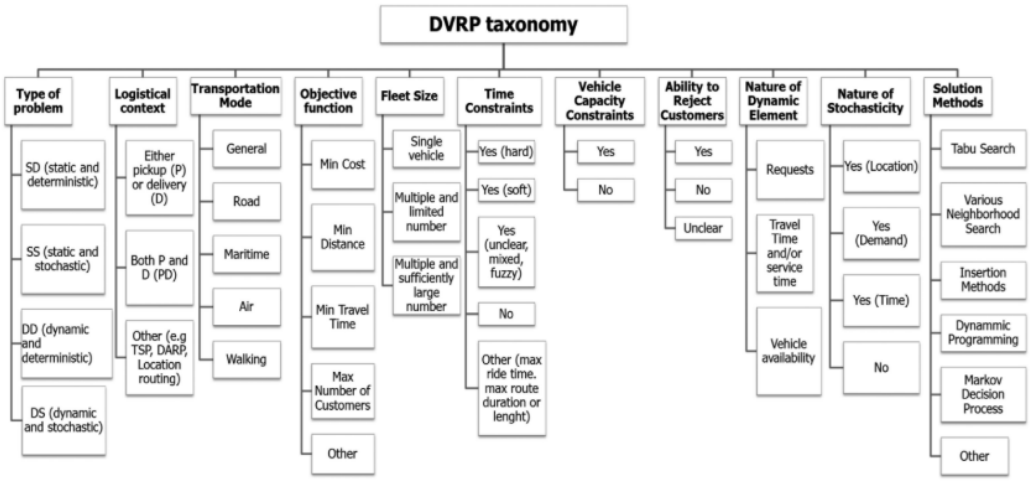


Figure 2.2: Taxonomy on DVRP, proposed by Psaraftis (2016) [43]

Concerning DVRP and its dynamic properties, new constraints might appear while a certain operations plan is being executed. The model has to have the adaptability characteristics to adapt and attach new constraint. A demonstrative yet straightforward example is vehicle breakdown. Upon breakdown, the system can no longer rely on that vehicle to accomplish more requests, meaning the fleet size constraint needs to be updated.

2.5 Taxonomy on DVRP

An important contribute to DVRP was made by Psaraftis et al. [43]. The work classified several papers on the subject. It classified them using a proposed taxonomy with 11 criteria: type of problem, logistical context, transportation mode, objective function, fleet size, time constraints, vehicle capacity constraints, the ability to reject customers, the nature of the dynamic element, the nature of the stochasticity (if appliable), the solution method. Although not wholly independent, the 11 criteria provide an overview of the different approaches and constraints published in the topic of DVRP (Figure 2.2).

2.5.1 Type of Problem

A VRP routing problem can be Static and deterministic (SD), Static and stochastic (SS), Dynamic and deterministic (DD) and dynamic and stochastic (DS). This taxonomy used the definition of "dynamic" proposed by Psaraftis (1988) et al.[42]. Psaraftis classifies a VRP as dynamic if the problem inputs are continually received simultaneously with the route calculation. Contrarily, a problem is classified as static if all the inputs are received forehand and do not change during the routes' execution. A problem is considered stochastic when its inputs are not known with certainty and deterministic otherwise.

2.5.2 Logistic Context

This criterion classifies the nature of the problem. VRPs can be divided in Either pick up or delivery (P/D), Pickup and Delivery (P/D), Routing with Location/Inventory Considerations, and Routing with Queuing Considerations [43].

2.5.3 Transportation Mode

The transportation mode influences the logistics of the VRP. The proposed taxonomy describes four types of transportation modes referenced in the literature: Road, Maritime, Air and Walking. Road transport accounted for the most described problem [43].

2.5.4 Objective Function

The Objective function is referred to as a criterion of significant importance. The authors are critics about the objective functions described in the literature to solve DVRPs. Most objectives functions are classified as "identical or quasi-identical to traditional static problems" [43], with some exceptions. Two big sets of objective function clusters are defined: minimisation and maximisation objective functions. Some examples of minimisation problems on the literature are minimised route cost, route distance, travel times, total lateness, number of vehicles, cost of service and customer dissatisfaction. Two objective functions to maximise were documented: quality of service and total profit. The authors propose the use of "throughput" and "per unit time" metrics; metrics such as average unit time serviced customers, average time cost per unit, average client rejections per unit time. They also suggest more complex objective functions that assign a higher weight to close events compared to later ones [43].

2.5.5 Fleet Size

The fleet size respects the number of vehicles available in the fleet. Problems are classified in single-vehicle, multiple and limited numbers of vehicles and infinite vehicles when their amount is sufficiently large. The most common problems describe a fleet with multiple but limited vehicle amount [43].

2.5.6 Time Constraints

Time constraints criterion concerns the type of time constraint in the requests and can be classified in No time constraints, Hard time window, Soft time window or another type of time constraints. In the case of DVRP, time constraints either do not exist or are considered soft. Soft time constraints are characteristic from problems were earliness or tardiness is penalised in the objective function. Hard time windows on DVRP might make a problem instance infeasible as they need to be respected even with disruptive events. Other types of time windows include time windows dependent on the customers' type, maximum ride time constraints or maximum route duration constraints [43].

2.5.7 Vehicle Capacity Constraints

Most papers incorporate vehicle capacity constraints. On the other hand, problems where the volume of goods is sufficiently small compared to the carrier vehicles' capacity, can be considered of infinite capacity. Problem variants, where a service is provided, instead of a physical good, can also be considered infinite capacity. Infinite capacity means that any vehicles can serve any number of customers [43].

2.5.8 Ability to Reject Customers

DVRP dynamic properties and associated uncertainty, usually presuppose the ability to Reject customers. The taxonomy's authors emphasise to the connection of this criterion with the time constraints criterion. The presence of hard time windows usually means that one can reject individual customers under the penalty of making the problem unfeasible. Depending on the problem, customers can also be rejected if the cost to include them in the operations plan is too large [43].

2.5.9 Nature of Dynamic element

Dynamic elements are dynamic inputs that are unpredictable and will cause disruption of the pre-calculated operations plan. This criterion classifies the different types of dynamic inputs in routing problems: Dynamic customer requests, with customers altering or cancelling requests, and changing their location on the network; dynamic travel or service times, influenced by road congestion or closed nodes on the network of roads; Dynamic vehicle availability or vehicle breakdowns [43].

2.5.10 Nature of Stochasticity (if any)

This criterion is only applicable to the DS and SS problems. Concerning its nature, stochastic events can involve Stochastic customer locations, Stochastic demand quantity, Stochastic travel times [43].

2.5.11 Solution Methods

As previously mentioned, on DVRPs, the time to obtain a solution is usually scarce, requiring the use of fast solution time methods, such as heuristics-based approaches. Most of the methods used to solve DVRP are the equivalent or adaptations of the ones used on solving static VRPs. Possible solution methods are Tabu Search, Neighborhood Search algorithms, Insertion Methods, Nearest Neighbor, Column Generation, Genetic Algorithms, Ant Colony Optimization, Particle Swarm Optimization, Waiting-Relocation strategies, Markov Decision Processes and Queuing-Polling Strategies. The taxonomy authors consider the four last methodologies with the tag of "more dynamic" compared to the others. Remaining methodologies are just adaptations of the ones used on static approaches. Concerning solution methods, the authors refer that the worst performance case analysis has not much presence on the documented DVRPs. This metric calculates the maximum distance a particular heuristic result can be to the optimal solution [43].

2.6 Measuring the Dynamism

As before mentioned, distinct problems encompass dynamic elements of different natures. Dynamic elements can also be characterised in two dimensions propose by Ichoua (2007) et al. [25]: frequency of changes and urgency of requests. Frequency of changes pertains to the rate at which the information is updated and becomes available. The urgency of requests concerns the time difference between acknowledging a new request and the time expected to fulfil it [38]. To measure an individual system's dynamism, Lund (1996) et al. [30] proposed a simple metric defined as the degree of dynamism (δ) (Equation 2.1). It encompasses the ratio of dynamic requests (req_{dyn}) relative to the total amount of requests (req_{total}) [**?**]. The simple metric proposed does not consider the times of the dynamic requests. Systems that receive a request at the beginning of the operations plan are perceived as equivalents to the ones who received the request at the end of the planning horizon [38].

$$\delta = \frac{req_{dyn}}{req_{total}} \tag{2.1}$$

Based on this problem, Larsen(2001) [28] proposed the effective degree of dynamism, an average of the disclosure times. As Larsen concluded, measures that seem promising for describing one system's dynamism might be inappropriate for other systems. Therefore, Larsen proposed the former metric for both problems without time windows ($edod$) and problems with the presence of time windows ($edod_{TW}$) [28].

2.6.1 Absence of Time Windows

Assuming that the requests received before the beginning of the planning horizon have a disclosure time of 0; the planning horizon ends in time T; T_i is the disclosure time of the request i; R is the set of requests; n_{imm} is the total number of requests received in the planning horizon.

$$edod = \frac{\sum_{i=1}^{n_{imm}} (\frac{T_i}{T})}{req_{total}} \tag{2.2}$$

2.6.2 Time Windows

This metric was extended to problems with time windows. The temporal difference between the time when a request is received (t_i), and the latest time it should be satisfied (l_i) is classified as reaction time. The metric beneath reflects that a planner would prefer requests with longer reaction time as they give more room to insert other dynamic requests in the routes [28].

$$edod = \frac{1}{req_{total}} \sum_{i=1}^{req_{total}} (1 - \frac{l_i - t_i}{T}) \tag{2.3}$$

Several other factors that affect the system's dynamism are not captured by any of the above metrics: geographic distribution of the requests and travel times between the accomplishment of requests [38]. The frequency of information updates has a significant impact on the time window available for the optimisation [38].

2.7 Problem Formulations

The literature describes several DVRP formulations, depending on the application context and the flexibility allowed in the revised operations plans. On its survey, Eglese, enumerates several essential considerations to be taken into account on Disruption management problem formulations [15].

Generally, in DVRP documented in the literature, goods are transported to customers at their demands. Vehicles are assigned with routes to fulfil customers' demands in the most optimised way possible, bearing in mind the dynamic environmental factors. The commodities to be transported might be specific to individual customers, for instance, in the case of package delivery companies. In this case, each customer can only be served by the vehicle that carries the commodities he demanded. Contrarily, if the commodity is generalised to all the customers, any customer can be served by any vehicle [15]. These constraints are vital as they maintain the problem's feasibility. In the former case, when vehicle breakdown occurs, other vehicles must visit the breakdown vehicle to pick up the individual goods and deliver them to the specific customers.

A second essential consideration mentioned by Eglese [15] is the degree of flexibility allowed when calculating the revised plan. Some formulations imply that upon vehicle breakdown, the remainder of the breakdown vehicle's route must be completed by a singular vehicle. However, as pointed out by the authors, there might be room for further optimisations; the remainder customers can be distributed by multiple vehicles. The order and priority of the customers that remain to satisfy also needs to be taken into account. Some formulations imply that this order is maintained once disruption occurs, while others allow the optimal reorganisation of customers [15]. A certain system might not desire this further optimisation, as some clients may need to be served with more urgency than others.

Another important formulation detail is to specify when the vehicles can diverge from the initial pre-calculated routes. In some formulations, the vehicles are required to finish their whole schedule, meaning they must serve all the customers previously defined before they can be used to tackle disruptions. Other problem formulations allow vehicles to be rerouted during the execution of an operations plan. Depending on the communication methods between the fleet vehicles and a centralised control centre, some formulations forbid vehicles to be redirected in the middle of a trip's execution (trip is the journey between two nodes on the road network). If the communication between the vehicles and control centre is continuous, it is possible to reroute a particular vehicle in the middle of a trip [15].

The last two considerations are related to the objective function. It is crucial to understand if new vehicles and their respective crew are available at a depot to be allocated by the system. In the presence of vehicles or crew shortage, the system should append the costs of hiring more capacity to the objective function [15]. As enunciated by Psaraftis on their taxonomy classification on DVRP [43], the ability to reject customers is an essential factor. Some formulations allow client requests to be rejected when it is not profitable enough to visit them. Other formulations imply that, even in case of disruption, all clients must be fulfilled. In this case, the system may serve clients in a schedule utterly different from the one initially calculated.

2.8 Solution Methods

As a priori mentioned, vehicle routing problems (VRP) are unanimously classified as NP-Hard. Therefore, DVRPs are expected to be also of NP-HARD complexity [56]. Some papers contain shreds of evidence of their formulations to be NP-HARD [15]. With few exceptions in the literature, DVRPs are always solved using approximations or relaxing the real constraints from the real-life problem. The planning horizon concept describes the time-window of the problem, since the moment the first vehicle departures from the depot to the instant where the last returning vehicle arrives at the depot. Regarding DVRP, essential and useful information is continuously being inputted to the problem as the progress on the planning horizon evolves. The complete instance of the problem is only known at the planning horizon's finale. The literature documents several solution approaches regarding DVRP. The solution methods available can be segmented two clusters [38], representing the two types of dynamic vehicle routing problems defined in taxonomy [43]: dynamic and deterministic routing problems and dynamic and stochastic routing problems.

2.8.1 Dynamic and Deterministic VRP

On the category of dynamic and deterministic VRP, the literature is divided into two possible solution approaches: Periodic optimisation and continuous optimisation [38].

2.8.1.1 Periodic Optimisation

The periodic optimisation approach is based on the segmentation of the planning horizon in small segments that are analysed and solved as a static VRP. The approach was first used by Psaraftis in a problem without stochastic information [41]. The dynamic problem is divided into multiple static instances, obtaining the most optimised routes in a particular timestamp. The static VRP is solved every time a new disruption occurs or in predefined time intervals. This approach suffers from the curse of dimensionality as it becomes too complicated and computationally costly for large problem instances. Since the optimisation process optimises the routes from scratch, the computational time will delay the communication of the revised plan to the vehicle fleet, ultimately causing the problem to be unfeasible. Periodic optimisation has the clear advantage to allow the use of algorithms already intensively studied and developed in the static VRP context [38] . A paper by Montemanni proposed an developed an interesting use of this approach, to solve a DVRP using Ant Colony Systems (ACS) [35]. They divided the planning horizon (a day) in multiple slices of the same duration, ensuring equal computational effort for each time slice. At the beginning of each time slice, the problem is solved with the available information at that moment. Any request received during that time is ignored and will only be accounted for in the next time slice. The requests studied in this paper were not urgent and allowed for the use of the former methodology. They also proposed and developed an interesting enhanced to periodic

optimisation, transferring promising solution characteristics between a time slice and the next. The approach was used in a realistic case study [35].

2.8.1.2 Continuous Optimisation

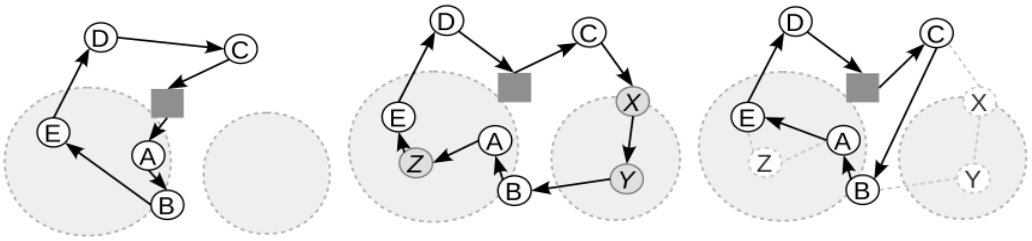
Continuous optimisation is done continuously during the day upon a disruption. This approach was first used by Gendreu (1999) et al. [18] with an adaptation of the parallel Tabu-Search algorithm to solve a Dynamic VRP with time windows, motivated by courier services. The current problem solution is maintained in memory, being updated continuously once the available problem information changes. A major disadvantage of this approach is due to the fact the routes are being updated continuously, meaning each vehicle's destination is only known once it finishes the current request. This might be incompatible with the way some systems operate. Several implementations of continuous optimisation in the context of dynamic customer requests are based on a route or solution pool. The pool contains a set of alternative solutions that are maintained in memory. It is initially populated with a feasible solution, contains all the routing plans. When the system receives a new customer request, it calculates if the request can be served. If the system accepts the request, it appends it to the solution pool and solutions that are not compatible are removed from the pool. A pool-update mechanism is responsible for guaranteeing that all the pool solutions are coherent with the problem's current state [38].

2.8.2 Dynamic and Stochastic VRP

Regarding Dynamic and stochastic VRP, possible solution approaches to track the problem are divided into two categories: the ones based on sampling and those based on stochastic modelling [38] .

2.8.2.1 Sampling

Typically, a DSVRP has few random variables that influence the decisions taken during the planning horizon. Sampling methods are based on the generation of new scenarios with distinct values for these variables. Information like the probability of a certain customer to appear on a certain area might be known prior. In this case, the algorithm can instantiate several customers on that area according to the known stochastic information and adapt the routes to serve the *dummy* customers. By adapting the routes to the *dummy* customers, one may think the system is losing its optimality, as the routes have to consider customers that do not exist in the real problem. That is true, but as the routes lose their optimality to the original static problem, they grow adaptability characteristics to dynamic events [38]. In DVRPs, a robust solution needs to be assessed in dynamic environments, meaning the algorithm does not know all the information prior to the plan's execution. As the routes are calculated based on the stochastic information about problem variables, the system will react better in dynamic events. Sampling suffers from the curse of dimensionality as the number of scenarios needed to reflect the reality can be too large [38]. The following example in Figure 2.3 can be used to illustrate sampling [38]:



(a) Initial Optimal Plan

(b) Optimal plan with *dummy* customers

(c) Optimal plan, removing *dummy* customers

Figure 2.3: Sampling examples, adapted from [38]

This example concerns a DSVRP with unknown customers location. Initially, an optimised route plan is generated based on a particular objective function. The optimised plan is: *A*,*B*,*E*,*D*,*C*. There is information available about where the customers are more likely to appear, represented by the shades in Figure 2.3. It is important to note that this plan completely ignores one area where customers are likely to appear. With available stochastic information, *dummy* clients *X*, *Y* and *Z* are instantiated in promising areas. A tour to serve the dummy clients is calculated: (*C*, *X*, *Y*, *B*, *A*, *Z*, *E*, *D*). Finally, the dummy clients are removed, and the solution becomes: (*C*, *B*, *A*, *E*, *D*). Although sub-optimal for the initial static problem, this solution is more adapted to the zones where customers are likely to appear dynamically.

2.8.2.2 Stochastic Modelling

Approaches based on stochastic modelling integrate the stochastic knowledge analytically. Although they better capture the stochastic nature of the problem compared to sampling, they have very complex formulations [38]. The work done by Powell (1988) used stochastic modelling, formulating a truckload problem as a Markov Decision Process (MDP) [40] [38]. Other examples of this approach are the use of Approximate Dynamic Programming(ADP) and Linear Programming algorithms adapted to dynamic and stochastic settings.

2.9 Performance Evaluation

Deriving from their complexity compared to static VRP, DVRP requires different metrics to obtain a foolproof method's performance. On his review on dynamic vehicle routing problems[38], Pillac, points out two metrics that can assess a method's performance: competitive ratio and value of information. Sleator and Tarjan (1985) first introduced the competitive ratio on their systematic study of online algorithms. The suggested metric compares the performance of an online algorithm to an optimal offline algorithm [50]. An offline algorithm can access all the information, both static and dynamic, available at the beginning of execution. Antagonistically, an online algorithm receives its information piece-by-piece during execution time; the entire input set is not available from the start[38]. An online algorithm can only perform as good as the offline version of the problem. An online algorithm is classified as best-possible if another algorithm does not

exist with lower competitive ratio [7].
Let P be a minimisation problem with the commonly used cost of the amount of time needed to visit all the desired locations[38]. Let $C_{off}(P)$ be the cost of the optimum offline algorithm's solution and $C_{on}(P)$ the cost of the online algorithm's final solution. Regarding the metric, an algorithm is said to be *c-competitive* (with a competitive ratio of c), if there is a constant α that satisfies the Equation 2.4

$$C_{on}(P) \leq c \cdot C_{off}(P) + \alpha \tag{2.4}$$

The main drawback of the competitive ration is the complexity and difficulty in proving the inequality on real-world problems[38]. Another metric, the *value of information* proposed by Mitrovic-Minic(2004) [34] is more flexible and practice for measuring the effectiveness of a DVRP solution method[38]. This metric compares the solution returned by a particular algorithm on an instance of a dynamic problem, with the solution obtained by the same algorithm on a static instance of the same problem (with the dynamic information known beforehand). Let x_d be the optimal solution of the dynamic instance of a problem P and x_s the optimal solution of the static instance with the same objective function f. The *value of information* (V), when using the algorithm α, is defined as Equation 2.5

$$V_\alpha = \frac{f(x_d) - f(x_s)}{f(x_d)} \tag{2.5}$$

The above metric (Equation 2.5) measures an algorithm's performance based on empiric results and does not require calculating optimal solution for the offline instance with other algorithms[38].

2.10 Benchmarks

Heretofore, very few benchmark problems were published and referenced for benchmarking of a specific DVRP algorithm. As referenced before, DVRP presuppose that decision have to be taken in real-time, meaning there is a small time-window available for optimisations. This restriction recalls the use of methodologies and approaches that demand less computation power. The usual approach is to attest a specific method's validity by comparing it with a higher computational cost algorithm. This approach was used in a problem where vehicle breakdown might occur during the execution of a pre-calculated operations plan, a variation of the Team Orienteering Problem (TOP) [33]. They propose a solution based on a heuristic. In this case, a more computationally expensive Genetic algorithm was used to set the heuristic benchmarks.

Other papers, such as Chen(2006) [11], adapt the benchmarks published by Solomon (1987) [51] for static VRP instances. The fifty-six problem benchmarks created by Solomon were adapted and used to demonstrate and attest the proposed model's validity [11].

2.11 Problem Variations

DVRPs variations are similar to the Static VRP, but they consider some unpredictable elements that influence the vehicles' routes in real-time. In a real-world scenario, these events are in the limit, infinite, as there is a panoply of factors that can more or less influence the vehicle routes. As expected, the literature adapts the infinite dynamic problem and only addresses the elements useful in their specific context. DVRPs variations must be seen as a base prototype model, as a considerably amount of scenarios conjugate several models to propose the most adapted formulation. For this purpose, the current section will approach the most common dynamic vehicles routing problems and give examples of specific problems with a set of disruption types that will be valuable to this work.

2.11.1 Dynamic Travelling Salesman Problem (DTSP)

The dynamic travelling salesman problem (DTSP) is the simplest variant of a DVRP [23]. The TSP consists of discovering the most optimised routes a salesman needs to perform to visit a set of cities. All cities must be visited precisely one time. The salesman must also return to the initial city, which is usually called depot in case of VRPs. The DTSP is inserted in the dynamic and stochastic VRP category, and it is considered to be of NP-Hard complexity. This problem is defined on a graph, where the travelling times between nodes is previously known. Customers demands are the dynamic element, as they arrive at each node following a Poisson of mean arrival rate (lambda). The salesman serves these demands, spending a fixed time per demand on each node [43]. In cases where (lambda) is extremely low, the DTSP is equivalent to a median problem, where the optimal travelling time can be achieved using the policy to move a vehicle to the graph's median, anticipating the next demand [23].

2.11.2 Dynamic Vehicle Routing Problem with Time Windows (DVRPTW)

In a real-life scenario where a company does deliveries or other services to a set of customers, there are always time-windows associated. Time-Windows commonly refers to the desired time to visit and satisfy customers requests. The difference between this problem and its static version, the Vehicle Routing Problem with Time Windows(VRPTW), is incorporating information received dynamically in the decision process. The time window defined to an individual customer can be hard or soft, as explained in section 2.5.6. Taniguchi [53] studied this problem with the dynamic element being the traffic intensity. A set of trucks had to visit particular customers in the predefined time windows. Soft time windows were specified, and earliness or tardiness in requests was penalised in the objective function. They proposed a dynamic formulation of the problem and compared it with a static version, where the traffic intensity was forecasted. Better solutions were obtained in the DVRPTW formulation. Archetti [3] addresses a problem where customers are the dynamic element and arrive in real-time to the system. This formulation allowed some requests to be denied, incurring on penalty similarly to when the time window is violated. Pan [37],

decomposed a DVRPTW in several static problems with the objective of maximizing customer satisfaction. In this problem, vehicles that leave the depot have knowledge of static customer requests to be served and routes are initially calculated and optimized for those requests. The dynamic elements of this problem are customer requests, that can be added during the execution of the optimal precalculated paths, and also the initial static customers' orders that can be cancelled. Again, serving a customer outside the desired time window will mean a penalisation.

2.11.3 Multiple Depots Dynamic Vehicle Routing Problem (MDDVRP)

This problem variant differs from others in the starting positions of each vehicle. Typically, DVRP recognises a single depot from where all the vehicles departure. Concerning MDDVRP there are several depots, to whom the vehicles need to return to replenish resources or after accomplishing the routes. Fang proposed implementing a multi-objective, multi-depot, multi-type, dynamic vehicle routing problem (MOMDMTDVRP) to manage the routes of emergence crews upon Geological disasters [16]. This formulation had multiple depots to serve customers spread in a large area, in this context resource centres and disaster areas respectively. Each disaster event appears dynamically and is given a priority according to the disaster severity. The solution approach was based on a hybrid ant colony optimisation minimising the time taken to transport resources to the disaster sites while respecting the disaster priorities (serve the higher priority customers first).

2.11.4 Dynamic Capacitated Arc Routing Problem (DCARP)

Dynamic Capacitated Arc Routing Problem (DCARP) is a problem widespread in real-life scenarios. A set of arcs (between two nodes in the roads graph) has an associated demand and needs to be served by a fleet of vehicles, ensuring that the routes' demand does not exceed the vehicles' capacity. The vehicles are available at a single depot and have a finite capacity. The classic version of the problem has typically two costs associated with each arc, one for serving it and another for travel purposes only [52]. An example of a scenario that describes this behaviour are the snowploughs. When these vehicles remove snow from the roads, they have lower speeds, leading to a higher cost of travelling (service cost). However, they might need to use an arc just for travelling purposes, moving faster and having a lower cost associated. Several dynamism sources to this problem were explored in the literature: the appearance of new demands dynamically, dynamic graphs, changes in the edge costs and vehicle breakdown.

Padungwech studied the influence that the number of iteration in a Tabu search algorithm has on a DCARP solution [36]. Their problem had the objective of minimising the travelled distance, but also the service times. The optimal policy to minimise only the travelling distance is simple, just wait until the end of the planning horizon and solve the static problem (after all dynamic events had happened) [36]. This DCARP required all dynamic requests to be fulfilled. The number of iteration of the tabu search algorithm did not improve the solution, in opposition to a more frequent schedule update and the method use to integrate new tasks.

DCARP can also be time-dependent when the start time of service determines the service costs on each arc. An example is a work by Tagmouti [52], addressing a DCARP with dynamic service times in each arc due to weather report updates. Their solution is dividing the DCARP into new static problems each time an information update is received. This approach is then compared to a static version of the problem where the dynamic elements are known to prior. The dynamic approach has better results since both problems were tested in dynamic settings.

2.12 Summary

DVRP are problems of major importance in any vehicle routing logistic context subject to internal or external factors that might influence vehicle routes. These factors can cause the planned routes to become unfeasible or sub-optimal at a certain point, which can incur costs or profit losses to the companies. Dynamic environments raise a set of concerns to the routing system, as every business context has its constraints, and dynamic factors might rise the need to soften them. A common approach is to soften the constraints and penalize them on the objective function when they are not totally fulfilled. This set of rules is problem-dependent, and some constraints cannot be relaxed, for example, in ASAE's context, the economic operators need to be visited in their work schedule.

DVRPs are considered NP-hard problems, meaning that exact methods are not promising in achieving good results in a reasonable time. The literature mainly uses heuristic-based methodologies, aiming to get the best solution in the shortest time possible. The time taken to obtain a solution is of significant importance on DVRP because the vehicle fleet cannot stop working to wait for the system to calculate the new solution. Dynamic solutions must be achieved and communicated to the fleet in very short time-windows, close to real-time. To achieve such results, the system should initially calculate a provisional static operations plan that will be executed and revised in real-time once disruptive events occur.

It is also essential to consider the use of stochastic information when it is previously available, allowing the system to have some degree of prediction of disruptive events. The initial static plan can be modified and adapted to tackle a disruptive event by accounting for stochastic information, minimising its impact.

Chapter 3

Algorithms

This chapter will review the state-of-the-art concerning the four optimisation algorithms used in this dissertation: Hill Climb, Simulated Annealing, Tabu Search and Large Neighborhood Search. Along with a brief definition of each algorithm, this chapter also provides insights on several approaches used in the literature combined with improvements to each base algorithm.

3.1 Introduction

As previously analysed, DVRP is an optimisation problem with NP-hard complexity. There are three main approaches to solving this problem: exact, heuristic, and meta-heuristics approaches. Depending on the complexity of each problem instance, the feasibility of each method can change, and they have advantages and disadvantages in different aspects. Each method needs to be adapted to the specific problem and analysed in several metrics, such as the quality of the solution found, the time taken to achieve a solution, and the memory allocated during the search process. All these approaches were intensively studied and used in real-life scenarios and refined to each problem they solved. Some literature works compared several methods using a set of defined metrics, but besides some conclusions over the respective problem instance, there are no conclusions of which of the methods is the best overall.

3.2 Solution Methods

3.2.1 Exact Methods

Exact methods grant they find the best solution after they finish their execution. These methods usually visit all the problem states, and deliberate all the possible solutions. Sometimes they crop some areas in the search space where the best solution is granted absent.

This approach is often not suitable for more significant problem instances, with a lot of customers or an extensive vehicle fleet, because the solution space is exponentially ample, and the computational times become too substantial. Since this dissertation modulates the problem as a

DVRP, the computational times become even more important as the solution needs to be provided in close to real-time, as the brigades are working and need valid routes.

The approach implemented by Barros [27] to solve the static instance of the problem described by this work, analyses the performance of the branch and bound algorithm. This algorithm explores all the branches with possible solutions and interrupts the exploration of a branch if it originates an unfeasible solution. This algorithm was implemented with a time execution limited to 5 minutes and could solve problem instances with 24 economic operators for one brigade or 14 economic operators for eight brigades. These limitations are far from the real problem instance and confirm that exact methods are not suitable to solve complex VRP problems. The DVRP variant of this problem has increased complexity, mainly in the utility function. Therefore, it is expected to be prohibitive for using such methods.

3.2.2 Heuristic Methods

In contrast to exact methods, Heuristic Methods don't grant to find the best solution for the problem instance. This method doesn't need to be optimal or rational but rather enough to reach an approximation to the global maximum of the problem. Heuristic methods are usually designed when exact approaches fail to deliver a solution in acceptable time intervals or to find an approximate solution to a problem instance when accurate methods fail to do so. A heuristic is a function that calculates the utility of each search branch and, based on the available information, guides the search throughout the most promising branches. The main objective of the heuristic is to solve a certain problem in a time window that is reasonable, obtaining a solution that is arbitrary close to the optimal. The literature describes trade-off criteria to decide when a problem should be solved using a heuristic approach: Optimality, Completeness, Accuracy and precision, Execution time. The Optimality criterion concerns problems with multiple solutions and if it is necessary to find the best solution to solve the real-lie problem. The second criterion is completeness, and when a problem has multiple solutions. In some scenarios it is demanded to get all the solutions, while in others, only the best solution found matters (VRP case). The accuracy and precision criterion deliberates if a certain heuristic can provide a confidence interval for the solution found and if the error compared to the optimal solution is supported in the real-life problem. The last criterion regards the execution time. Different heuristics have different converging curves, some converging faster than others; the balance between a fast converging heuristic and one that grants a better solution is often a step of major importance when solving an optimisation problem.

3.2.3 Meta-Heuristic Methods

Similarly to the heuristic methods, these methods can find solutions arbitrarily close to the optimal for significantly large problem instances (lots of customers and brigades). Meta-heuristics have a substantially better precision and find better solutions when compared to heuristics methods in similar time intervals. Metaheuristics search a large subset of solutions by sampling using a certain methodology since this subset is too extensive to be explored entirely [9]. These algorithms

guide their search, tendentially exploring the most promising areas in the search space, finding solutions in a reasonable execution time. Metaheuristics do not guarantee that the solution found is the optimal global solution or close to its utility. These methods are non-deterministic and are usually associated with stochasticity since the algorithm and the intermediate decisions rely on a set of variables that are randomly generated [8]. The same metaheuristics can be used in different problems with very few adaptations since they are not problem-specific.

Metaheuristics are classified according to several properties. Two essential properties are the type of search and how many solutions are stored concurrently. The type of search property ranks metaheuristics in local search or global search. Local search algorithms like Hill climbing are used to find local optimums, and they can't guarantee finding the optimal global solution. These algorithms commonly use a greedy approach, and they can get easily trapped in a local optimum. Global search metaheuristics improve the search and find better solutions, escaping local maximums and searching in the whole solution space. Global search methods include simulated annealing, tabu search, and iterative local search. The second main classification property regards algorithms based on a single solution or a population. Single solution algorithms improve and modify a single solution, while population-based approaches improve a set of candidate solutions. A population is a set of candidate solutions controlled by a group of variables that influence the size and the way the population evolves throughout the search. An example of this approach is the Genetic algorithms, and an example of a single solution algorithm is the simulated annealing.

3.3 Algorithms

3.3.1 Hill Climbing Algorithm

The Hill-Climbing algorithm (HC) is a local search method typically used in optimisation problems. It is straightforward to implement and requires only to store the current solution state. As with all the local search algorithms, HC fails to detect the unsolvability of a problem instance and, if there is no solution available, the search will continue indefinetely [47]. The algorithm starts with an initial solution and attempts to improve it iteratively as the search proceeds within a predefined running time. In each iteration, a new neighbour solution is generated based on the current one, and it is accepted if its utility is greater than the current best solution. Since worse neighbours are discarded, the algorithm cannot go across less promising zones in the search space and will get stuck in local maximums. This affects the algorithm precision, making it unlikely to find the optimal global solutions.

3.3.2 Simulated Annealing Algorithm

Simulated annealing is a stochastic approach inspired by thermodynamics and is an analogy with the annealing processes in metallurgy. These processes involve many efforts to control the annealing temperatures to make the crystals bigger and have fewer defects. Simulated annealing is a metaheuristic that performs a global search with a single-solution approach and can be used in

substantial problem instances, achieving a solution close to the global maximum with reasonable execution times.

Similarly to the Hill-Climbing algorithm, simulated annealing starts with an initial solution and generates new ones on each iteration. During the search, it stores two solutions: the candidate solution and the best solution found so far. The candidate solutions are analysed in a particular iteration and are given as input to a function that generates a new solution. The new solution might be accepted as a candidate for the next iteration if it has a greater utility value or a certain probability value. The algorithm also stores the best solution and updates it throughout the search; this solution is returned by the algorithm when it finishes execution.

Contrarily to the previous Hill-Climbing algorithm, simulated annealing tries to escape the local optima by allowing the acceptance of worse solutions, than can later conduce the algorithm to the most promising zones. A new move is accepted if its utility is better than the current solution's utility or with the probability P_a shown in equation 3.1, where t and δ_{util} the utility difference between the current solution and the candidate solution. Accepting worse solutions is done based on the current temperature of the system, being controlled by a cooling schedule that adjusts the temperature along the algorithm's executing time. Similarly to annealing processes in metallurgy, initially, the system has more energy, meaning it can accept worse solutions with a higher probability. During the algorithm execution, the temperature is lowered until it reaches a minimum or a null value, meaning the algorithm will only accept solutions with a higher utility than the previous one; this behaviour is similar to Hill-climbing. Any solution with a better utility is accepted during the whole search.

$$P_a = \varepsilon^{\frac{\delta_{util}}{t}} \tag{3.1}$$

3.3.2.1 Neighborhood Structure

A neighbour solution is one that can be obtained by applying an operator to a solution given as input. The neighbourhood size is variable and always depending on the number of operators available and their specificity. Typically in simulated annealing implementations, the neighbourhood size is defined at the start and remains unchangeable during the search process. Although not as important or studied in the literature as the temperature, the neighbourhood size considerably impacts the algorithm performance [57]. Larry [21] studied the impact of the choice of neighbourhood size on the algorithm's performance and the convergence speed. This paper studied a SA algorithm applied to an instance of the TSP with uniform city distribution where the minimum value of the cost function was known beforehand, running for a predefined number of iterations. Larry [21] uses the concept of k-optimality, which says that a route has the most utility when compared to all the routes composing the neighbourhood. Two routes are neighbours with a k value of N if they can be obtained using an N-opt operator. The previous means that one route can be broke in N or less different places, and after being connected, it can originate the other one. The study concluded that the neighbourhood size influences performance differently among the same

problem instance tested with different amounts of cities. The choice k=3 seems the most reasonable, considering all the scenarios and higher values of k for problems with a very high number of cities. Since the algorithm had a predefined number of iterations to execute, it can be concluded that the values of k that found a better solution are fasting the convergence of the method.

Another approach to the neighbourhood size is to dynamically adjust it during the algorithm execution, maintaining the best value for each search stage. Such an approach was studied and applied by Yao [57], where the temperature values control the current neighbourhood size. In the exploration phase, the probability of accepting a "bad" or "good" move is higher, and thus a large neighbourhood size is used to increase the exploration. When the temperature is lower or null, a smaller neighbourhood size becomes more appropriate to exploit a smaller search area which presumably contains a good solution. Yao [57] tested this approach in an instance of TSP with equal parameters, comparing it with a SA algorithm with fixed neighbourhood size. Preliminary results demonstrated the advantage of using a dynamic neighbourhood size during the search.

3.3.2.2 Cooling Schedule

Since the parameter of major importance is the temperature, the cooling schedule, which is responsible for manipulating the temperature, has been well studied in the literature with several methodologies proposed. According to Atiqullah [4] three factors generalise the structure of a cooling schedule: Temperature, Markov chains and Temperature decrement.

The temperature should be initialised to a high value that allows the algorithm to accept virtually any new solution, even solutions with lower utility. In this stage, every configuration can be accepted with a similar probability. The temperature must reach values close to zero or values low enough such that virtually no worse solution is accepted, guaranteeing the algorithm's convergence.

$$\vec{\Delta C} = \frac{\sum_{i=1}^{n_{attempts}} |\Delta C_{i,j}|}{n_{attempts}} \tag{3.2}$$

Atiqullah [4] describes an approach to calculate the ideal initial temperature, based on both cost increasing and cost decreasing moves. Using Equation 3.2, he estimates the expected change in the utility function, the average difference in utility from solution A to solution B, a neighbour of A. The element $|\Delta C_{i,j}|$ represents the absolute value of the difference in the utility function between the solutions i and j, and $n_{attempts}$ represent the number of neighbourhood configurations. It is expected that the several neighbourhood configurations follow a normal distribution pattern with a standard deviation of $\sigma_{\Delta C}$.

$$t_0 = \frac{(\vec{\Delta C} + 3\sigma_{\Delta C})}{\log(\frac{1}{\alpha_0})} \tag{3.3}$$

The formula used by Atiqullah to calculate the initial temperature is presented in Equation 3.3. His experiments show that this formula performs around 50% better than other adaptative schedules. The α_0 represents the acceptance ratio, the percentage of moves that are expected to

be accepted at the initial temperature. An acceptance ratio of 0.9 means 90% of the neighbours of the initial solution are expected to be accepted by the algorithm because they have higher utility or by the stochastic acceptance rule.

The Markov chains can have variable lengths throughout the search, and its length corresponds to the number of transitions attempted at a specific temperature value. A Markov chain is interrupted when the algorithm accepts a new solution, updating the candidate solution. The majority of the schedules adopt more minor temperature decrements both linearly and using complex functions. Significant temperature decrements require longer Markov chains since more transactions are needed to achieve the quasi-equilibrium state. It is essential to analyse the trade-off between the temperature decrement and the Markov chains' size because small temperature decrements slower the algorithm's convergence.

The work by Atiqullah [4] proposed a parametric cooling schedule that combines the simplicity of the static and non-adaptative schedules with adaptative schedules that are guided to specific problems. To understand the importance of a cooling schedule, it is essential to reflect on the three distinct stages of a SA: global positioning, local search, and refine the solution. A good cooling schedule should consider the previous three stages of the search. Initially, if the temperature drops too rapidly, the algorithm can get stuck in a local maximum. During the middle part of the search, the algorithm should slowly decrement the temperature and settle on a search area where a local maximum is located. In the third search stage, the temperature should be kept at a low value so that the algorithm can find the local maximum similar to a Hill-Climbing algorithm [3.3.1]. Atiqullah proposes the Gaussian-like temperature decrement function during the entire search process, as shown by Equation 3.4.

$$t_k = t_0 \times \alpha^{-[\frac{1}{f}]^b} \tag{3.4}$$

$$b = \frac{P}{Q} \tag{3.5}$$

$$P = \log\left(\frac{\log(\frac{t_0}{t_f})}{\log(a)}\right) \tag{3.6}$$

$$Q = \log\left(\frac{1}{f}\right) \tag{3.7}$$

The variable k is an increasing Markov chain counter; t_0 and t_f represent the starting and final temperatures, respectively. The constants A and F are tunable and control the algorithm's time in high or low temperatures. If it is desired for the algorithm to spend less time in high temperatures, the value of the constant A should be increased, or the value of F should be decreased. In his paper, Atiqullah proposes a cooling schedule that will reduce half of the initial temperature in around one-third of the number of Markov chains allowed. To achieve the previous, Atiqullah used $f = 1/3$ and $a = 2$.

The stopping criteria of a SA implementation can vary in the specific problem and can be as simple as a determined number of iterations or a limited execution time. In his proposed im-

plementation, Atiqullah defines the stopping criteria as a specific number of Markov chains. The algorithm also terminates if one of three other criteria is met; the main criteria are the first two, and the last is only used as a "last resource" stop condition. The first criterion terminates the algorithm if there is no utility improvement in a defined number of Markov chains. The second criterion ceases the search if the utility improvement after five iterations is below a specific value. The last criterion is regarding the number of Markov chains. The algorithm will stop if the first two criteria were not met and the maximum number was reached, which, according to the author, is a sign of insufficient annealing.

3.3.3 Tabu-Search Algorithm

Tabu search (TS) is a metaheuristic, first introduced by Glover [19], that implements a local search and can be used to solve complex optimisation problems. Local search methods tend to get trapped in suboptimal regions as they accept only improving moves. Similarly to simulated annealing, tabu search relaxes this rule and can accept worse moves when no improving move is available. When the solution reaches a local maximum, the search doesn't get stuck because the algorithm then starts accepting worsening moves that will "unlock" new promising search regions. The tabu search also has a prohibitive rule, not allowing moves to be repeated for a defined number of iterations, discouraging the search to come back to solution regions already visited.

3.3.3.1 Tabu Moves

TS uses memory structures to keep track of the already visited solutions since they don't need to be revisited, or the search would be less efficient and could get trapped in a loop. Ideally, and in theory, the algorithm would store all the solutions in memory and compare them to the new ones. This approach can't be used in most real-life problems as the number of possible solutions is exponentially larger proportionally to the problem's complexity (more vehicles and more cities to visit) and would take too much memory space. Also, the computational time spent to verify if a new solution was already explored is prohibitive to the algorithm's execution. The solution to these problems is to find alternative ways, tabu criteria, to approximately represent the recently visited solutions and avoid returning to them for a predefined number of iterations.

Tabu tenure is the number of iteration that one move will remain tabu. The tabu list is a list that contains all the tabu moves and their respective tabu tenures. To easily the implementation, the memory structures store the iteration where a move will stop being tabu, so the memory doesn't have to be updated at each iteration. It is recommended to have tabu tenures that are dynamically adjusted as the algorithm executes. A work by Wassan et al. [54] studied the use of a reactive tabu search algorithm (RTS) in a vehicle routing problem with pickups and deliveries (VRPPD). They dynamically adjust the tabu tenures based on two mechanisms that react with the repetition of solutions. The first mechanism increases the tabu tenure when solutions are repeated and reduces the search spaces that don't need long tabu tenures. The second mechanism diversifies the search when it is confined to a zone in the solution space. Wassan's approach performed better when

compared to previous literature's algorithms applied to benchmark problems, finding new best solutions for some of the problem instances.

3.3.3.2 Diversification / Intensification

Tabu search is often implemented with additional strategies to improve the basic algorithm. Glover et al. [20], in his guide to tabu search, describes intensification and diversification as "additional ingredients in order to behave as an intelligent search technique."

Intensification is a strategy based on the assumption that after visiting few "good" solutions, it is possible to relate them and find common properties. These properties can then be used during the algorithm execution to change the neighbourhood generation, favouring solutions that verify them. This strategy creates a bias and indirectly controls the solution structure according to such properties and discourages them from being violated. This strategy can use new memory structures, such as frequency-based memory that stores the frequency of specific solution attributes or moves performed during the search. Elite solutions or parts of them can also be stored and compared to the new solutions.

A basic implementation of the tabu search can get stuck in certain areas of the search place if the solutions are very similar. The diversification strategy tries to conduct the search to other unexplored areas by modifying the objective function while the algorithm is running. The utility function is adapted to favour solutions that contain less frequent attributes and penalise solutions with elements that appeared in other solutions visited. Another possible approach for diversification is the use of random restarts, where the algorithm is restarted using different solutions.

3.3.3.3 Aspiration Criterion

Since it is impossible to store all the solutions explored by the search, the moves that originated such solution become tabu and cant be used for a defined number of iterations. A tabu criterion is never exact, which means that some unexplored solutions might be considered tabu. Therefore, the algorithm might consider a solution that would improve the best solution as tabu, skipping it.

An aspiration criterion is a rule that allows the algorithm to perform a tabu move if this move is guaranteed to generate a solution that was never visited during the search. The most straightforward aspiration criterion is to check if a new solution is the new best solution. In this case, this solution was never explored. If all moves are considered tabu, an aspiration criterion can choose the one with the highest utility value, even being a tabu move. More complex aspiration criteria involve the solution's utility. When the tabu-list is updated with a new tabu move, the solution's utility generated by that move could also be stored in the tabu-list. Therefore, when a new solution is rejected for being tabu, the algorithm can check if its utility is similar to the one stored in the tabu-list. If these two utilities don't match, the new solution was never visited and should then be accepted, even though it is considered tabu by the tabu criterion.

3.3.4 Large Neighborhood Search

Large Neighborhood Search (LNS) is a metaheuristic proposed by Shaw [48] in 1998 that gradually improves an initial solution by segmenting and later repairing it. The LNS is typically scheduled to stop after a defined number of iterations have passed [46]. This particular heuristic belongs to the heuristic class named Very Large Scale Neighborhood search (VLSN) algorithms [39]. VLSN are algorithms that search large neighbourhoods, and it usually results in finding better local optima. Although this approach is expected to return better solutions, searching an extensive neighbourhood is often too expensive as both the time and computational power needed can be prohibitive.

A good approach, and the one used by LNS, is to restrict the search space to a size that can be efficiently searched. The LNS narrows the neighbourhood by a destroy and repair method. The destroy method destroys a portion of the solution in a stochastic way so that different elements composing the solution are destroyed in each method invocation, originating different solutions. The repair method is used to rebuild the solution. The neighbourhood of a solution is defined as the set of new solutions that can be reached after consecutively applying the destroy and repair method [39]. An example of a destroy method applied to a solution in a TSP instance can be randomly removing N customers from the solution and reconnect all the routes with the remaining ones. The repair method could use a greedy approach to re-insert these customers, starting with those whose insertion costs are lower. In a problem with 100 cities, there are $C(100, 15)$ ways of selecting the customers to remove and many ways to repair the solution for each of them. Different combinations of removed customers can result in the same solution after repairing.

The destroy and repair method can be interpreted as fixing and optimising operations [46]. The fixing part consists of sticking part of the solution, meaning it won't be changed or optimised, leaving a smaller portion of the solution. This smaller portion will be optimised by a defined method that will improve its utility without changing the components from the fixed part. Hopefully, the new solution generated after repairing will have a better utility compared to the initial solution. This search method uses three distinct variables: one storing the best solution found during the search process, another to represent the current solution and the last one stores a temporary solution that can be rejected or assigned as the current solution. The LNS is used with other heuristics that will do a local search in the new neighbourhood, allowing the heuristic to navigate quickly and easily throughout the solution space. After the heuristic returns the best solution found, the repairing method is called, generating a new solution. The new solution's utility is evaluated and accepted according to an acceptance function for the next LNS iteration.

3.3.4.1 Acceptance function

The literature described several acceptance functions, and the simplest one only accepts a solution if it has a higher utility than the current solution's utility. Always accepting the best solution can

cause the algorithm to get trapped in a local minimum [46]. Ropke et al. [46] proposes an acceptance function inspired by the SA where a worse solution can be accepted with a certain probability. The algorithm starts with an initial temperature, T_{start}, and the temperature is decreased linearly along with the iteration counter. Since the initial temperature is problem-dependent, their approach calculates it by analysing the initial solution and picking the value, leading to 50% of the worse solutions being accepted.

3.3.4.2 Destroy method

According to Ropke et al. [39], the destroy method is an important part of the LNS heuristic, being the degree of destruction the choice with more importance. If the degree of destruction is lower (a small portion of the solution is destroyed), the algorithm might have trouble when exploring the search space because a large section of the neighbourhood was lost. Antagonistically, when a large part of the neighbourhood is removed, the algorithm will behave similarly to a search in a large neighbourhood having the same computational time problems or give rise to poor quality solutions. To overcome these problems, the authors refer to two methodologies proposed in the literature: progressively increase the degree of destruction during the algorithm execution or choosing it randomly, in each iteration, from a range of defined values. The importance of selecting a suitable destruction method relies on the fact that all the search space should be reachable or, in the limit, the search space where the best solution is expected to be located. Hence, the selected method has to be capable of destroying different parts of the solution every time it's called.

3.3.4.3 Repair method

There is much freedom when it comes to select the repair method, and the first dilemma is choosing between an optimal that grants the best solution constructed from the partial solution or a heuristic method that tried to construct a good solution from the partial one [39]. While the optimal repair methods might be computationally slower than the heuristic ones, they usually provide better solutions in few iterations. Ropke et al. [39] also refers that optimal repair methods are less attractive from the diversification point of view as they will provide solutions with identical values for the cost function; the algorithm may get stuck in some valleys of the search space.

3.3.4.4 Adaptive large neighborhood search

The Adaptive large neighbourhood search (ALNS) was proposed by Ropke et al. [46] as a heuristic that extends the LNS for solving a VRPTW. This approach differs from the basic LNS, allowing different destroy and repair methods during the same search, called sub-heuristics [39] [46]. The sub-heuristics are selected with a frequency according to their past performance [46]. Using different sub-heuristics indicates that distinct neighbourhoods are being generated and chosen based on their past performance to find a better solution. After they finish execution and return a solution, a score value (weight-θ) is assigned for both destroy and repair methods. This score θ is computed using a formula with four distinct values determined by the utility of the solution returned by the

two methods: V_1 if the solution is a new global best, V_2 if the solution is better than the current one, V_3 if the solution was accepted and V_4 if the solution was rejected. The values are defined taking into account the inequality 3.8, and a higher value of θ corresponds to a more successful sub-heuristic. The distinct sub-heuristics are selected using the roulette wheel principle, and the weights associated with each sub-heuristics will define the probability of the respective heuristics to be picked. This approach will reward sub-heuristics that will progress in the search (find better solutions) and discourage heuristics that tend to generate rejected solutions and consume an iteration for no value add up.

$$V_1 \geq V_2 \geq V_3 \geq V_3 \geq 0 \tag{3.8}$$

3.4 Summary

The optimisation problem analysed by this dissertation is an instance of the DVRP and its of NP-Hard complexity, meaning it can't be solved exactly in polynomial time. For this reason, exact methods should be discarded, especially for more significant problem instances with many customers to visit or many vehicle routes. Exact methods cannot be used to solve the ASAE's routing problem, especially its dynamic version, as the inspection routes need to be updated in close to real-life. Instead, approaches based on meta-heuristics can be used since they allow a solution to be progressively improved over time. The algorithms keep a valid solution during the search that can either be further optimised or sent to the brigades operating in the field.

The Hill-climbing method is the most straightforward to implement and the fastest one of the four to converge to a solution. However, this method is the one more suitable to get stuck in the local maximum. The other methods accept lower utility solutions during the search and can escape the local maximum when correctly implemented.

Chapter 4

The ASAE Case-study

This chapter studies how economic and food safety is managed both in Portugal and on international context. The chapter also explains how artificial intelligence and state-of-the-art data mining techniques can optimise current processes in these two domains. In Portugal, ASAE is the government institution responsible for these two domains. Their system is currently becoming outdated and thus can be greatly improved on the context of Incode2030.[1] Further optimisation can be achieved by allying real-time information captured by nowadays devices with machine learning in data and text mining and artificial intelligence approaches.

This chapter will also review the system implemented in the purpose of Project IA.SAE, focusing on the route optimisation module. The proposed problem formulation and the utility function used to calculate each economic operator's utility will be reviewed. It will then provide an overall look at the system architecture and the algorithms used to solve the VRP. Finally, a brief description of the previous work's conclusions is provided.

4.1 Introduction

Economic and food security are two factors with significant importance in any developed country. The actual pandemic context of COVID-19 emphasised their importance [14].

Regarding food security, the European Commission implemented an integrated Food Safety policy in the European Unit (EU) to regulate the alimentary products "from farm to fork". These policies aim to protect consumers while guaranteeing the smooth operation of a single market.[2] Food safety describes operation from handling and preparation to storage of alimentary products, preventing contamination by agents that cause illnesses. These three domains must be present in

[1] Project Incode2030 - Metas, available at https://www.incode2030.gov.pt/metas, accessed on 2021-01-02

[2] Food safety - Europe, available at https://eur-lex.europa.eu/summary/chapter/food_safety.html?root_default=SUM_1_CODED%3D30&locale=en, accessed on 2021-01-02

all the processes since the product or its raw materials are cultivated until the final consumer consumes it.[3] Both national and international organisations have published norms and regulations to prevent behaviours and attitudes than risk public health. Several organisations, including ASAE in Portugal's territory, are responsible for inspecting the economic agents operating in Portuguese territory for non-compliance with the law. These organisations will benefit from the enormous amount of data collected nowadays that can be processed with machine learning techniques retrieving useful knowledge. The knowledge will be used by artificial intelligence approaches to generate systems capable of optimising the already implemented inspection processes.

Economic monitoring is the set of processes that monitors the economic agents' financial activities. This brings fairness among different economic agents operating in the same economic environment. Regulated business practices will also benefit the final consumer.

4.2 Food and Economic Security Authority (ASAE)

Autoridade de Segurança Alimentar e Económica (ASAE) is the Portuguese entity responsible for inspecting economic operators in Portuguese territory for non-compliance with the law. This institution was created in 2005 and resulted from the fusion of several departments previously operating separately. It is currently responsible for the areas of food safety and financial auditing. ASAE is part of the criminal police and remits to the Ministry of economics, assessing and communicating risk in the food chain and economic practices. It contemplates four areas of intervention: food safety, tourism and commercial practices, products and facilities safety, intellectual and industrial property.[4] On their plan of activities for 2021 [14], ASAE analyses the current pandemic situation as something with no parallel precedents. This situation arrives new challenges as the public health risk massively increases.

ASAE's organisation is composed of five nuclear and centralised units and three decentralised units (Diagram 4.1). The three last units have their operation encapsulated in regional areas and are subdivided in twelve regional Operation Units (4.2) [14].

4.2.1 Economic Agent Inspection

In the scope of this dissertation, closer attention will be given to ASAE inspection methods. ASAE currently has no intelligent system that can define priorities on which economic agents to inspect. Inspections are currently performed following an inspection plan calculated at the beginning of defined periods (three times a year). The national operations planning unit's precalculated plan and uses a series of macro-factors to assign priority to specific groups of economic agents. Specialists define sectors and activities of higher priority to inspect at the beginning of the year coordinated with the available logistic and human resources. As the priorities are only defined for sectors or activities, it becomes hard to plan which individual economic agents will prioritise inspections.

[3] Food Safety - European Commission, available at https://ec.europa.eu/food/overview_en, accessed on 2021-01-26
[4] Autoridade de Segurança Alimentar e Económica., available at https://www.asae.gov.pt, accessed on 2021-01-31

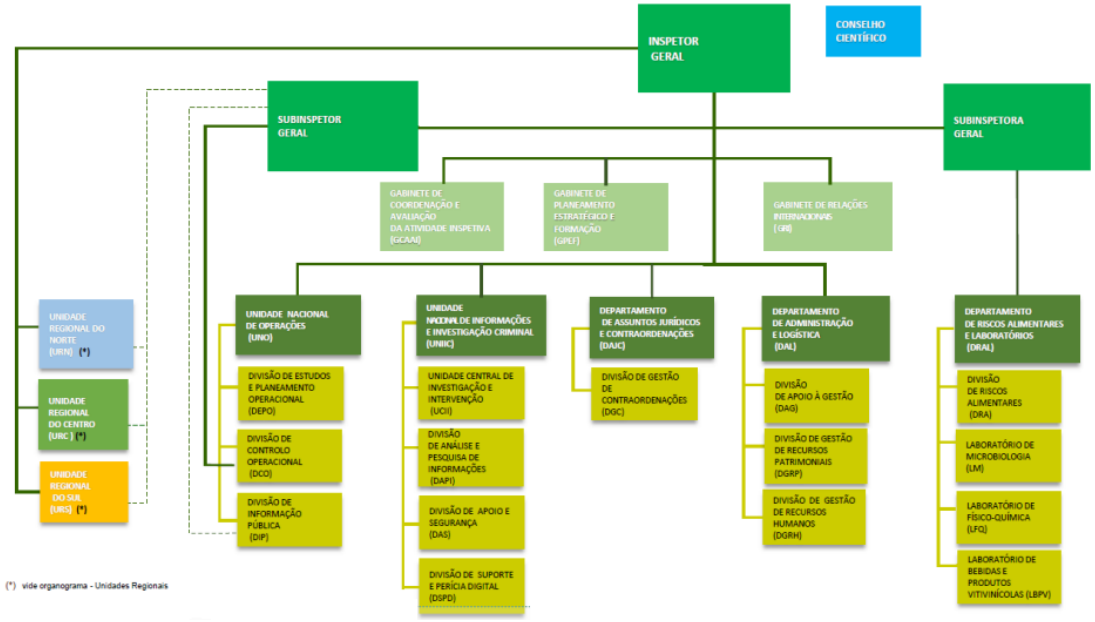


Figure 4.1: ASAE's organization chart

4.2.2 Optimising Inspections

Further optimisations to ASAE's inspections can be done in two different areas of action. First, by improving the utility of the economic agents to visit in a particular day and then by choosing the set of routes, vehicles and crew that are most optimised for a particular inspection day. Routes, vehicles and crew must be flexible and dynamically adapt to disruptions, characteristics of the dynamic environment operated by ASAE.

ASAE has databases that are continuously collecting data about their operations. This data is originated from their inspection reports and complaints written by consumers. Project IA.SAE, resulting from a cooperation between LIACC and ASAE, implemented means to process this data and extract valuable knowledge from it. State-of-the-art machine learning approaches, both in data and text mining domains, were used to process the data, finding patterns and extracting knowledge. The valuable knowledge was used to assign a utility value to economic agents, reflecting how promising they are to have non-compliances with the law. The utility function used to calculate the utilities was based on factors such as past complaints.

Project IA.SAE made substantial signs of progress on optimising the inspection routes. Routes are calculated maximising the total utility of the economic agents to visit while minimising the allocated resources for those operations. Total travel time, vehicle wear and human resources are examples of resources to be minimised when calculating inspection routes.

Figure 4.2: ASAE's Regional units

4.3 Big Data in food safety

> "The massive rise of Big Data generated from smartphones, social media, Internet of Things (IoT), and multimedia, has produced an overwhelming flow of data in either structured or unstructured format. Big Data technologies are being developed and implemented in the food supply chain that gather and analyse these data. Such technologies demand new approaches in data collection, storage, processing and knowledge extraction. [26]

In a world where collecting data becomes more accessible and at fewer costs, the opportunity to use this data to acquire useful knowledge is encouraging and promising. Big data is a field of studies that aims to analyse and extract valuable information from vast and complex data-sets which would be impossible for traditional data-processing software.

Although in an embryonic state, the application of big data and artificial intelligence approaches to Food safety domains is very promising and has gathered considerable attention in the past years. A review of Big data in food security by Jin, reviewed and categorised 113 relevant papers in this topic. "Data sources and collection", "big data infrastructure" and "data analysis" are three essential steps in the big data framework [26].

4.3.1 Data sources and data collection

Numerous and different types of data that can generate useful knowledge on the food safety domain are obtained from several sources such as online databases, Internet, mobile phones and social media [32]. There are some challenges associated with collecting and processing this data, for instance, with non-traditional data sources like social media. Social media data is not structured, and is mostly composed of free text, difficulting the knowledge extraction process [32]. There are also data in a highly structured format that can be easily extracted and stored in a database [26].

Platforms like Twitter, Facebook or Youtube are gathering the attention of the domain of big data. Questionnaires, public discussions, trends or opinions published on these social media can be seen as a valuable source of data for food safety [26]. A work by Chung [12], was one of the pioneers to empirically investigate the influence of a company's apologies during a crisis and its influences in online sentiments. This work analysed the evolution of online sentiments towards a company in a food crisis. A large data set of tweets was used (over 2.6 million) concerning a food poison case in 2015/16. Using a supervised machine learning approach learning from the Tweets dataset, they concluded that the company's apologies did not remove the public concerns about food safety caused by the crisis. Similarly, Internet webpages can provide essential data related to food safety, mainly using web crawling techniques [32], where a program can navigate the Internet autonomously and find relevant information about a subject.

Online databases are public databases that can be considered relevant data sources as they contain pieces of information related to food safety. Example of relevant information are threat alerts given by monitoring programmes and chemical analysis, exposure data gathered from consumer databases and reports on animal and pants diseases [32]. Smartphone sensors can also be a useful data source for several purposes like quality control, food inspection, behaviour management and communication of food safety-relevant information [32]. Satellite imagery can also be a fundamental data tool for assessing and improving products' quality, mainly on the cultivation phase [26].

4.3.2 Big data infrastructure

Supercomputing is necessary to process Big data and its respective challenges derived from its dimensionality and complexity [26]. European Union, United States of America and China have deployed several cloud computing and high computing infrastructures to process Big data, and they are also being used on the food safety area, supporting decisions that improve public health.[5]

4.3.3 Data analysis

Data analysis is the step where valuable knowledge is extracted from the data and is the core step in data processing [26]. Marvin [32] refers to two methods for Data analysis regarding big data on food safety: recommendation systems and machine learning.

[5] SAIC High Performance Computing Contributes to Healthy Lives, available at https://www.saic.com/features/high-performance-computing-at-FDA-keeps-foods-safe, accessed on 2021-01-05

Recommendation systems are a subclass of information filtering systems to predict the rating or preferences of a group or target user to a specific item. Organisations use these systems to advise their customers based on information such as best seller products, customers' location, and products bought by the customer [32]. However, not a very common approach in the food safety area, these systems were proposed by Singh [49]. A recommendation system based on Twitter data for issue detection in the supply chain management of food industries was proposed. This system allowed for decision making supported by customer feedback and reported issues in food products' quality [49].

Machine learning encompasses algorithms that learn knowledge from data. These algorithms can be used to build models that can predict or make decisions in the food safety area [49]. As the models are learned from vast amounts of data, specific patterns are captured, that would otherwise be invisible for the human interpretation. Several Machine learning algorithms are standard in the food safety context, such as Bayesian networks, Neural Networks Random Forests and Decision-Trees [49]. Machine learning algorithms are used in the food safety context to tackle problems in 3 different domains: food quality assessment and management, using computer vision and deep learning methods; identify, monitor and forecast risks in food safety; extract food safety information from digital text data [49]. An alarm system was proposed by Chang [10], using machine learning to extract valuable knowledge from nearly 100 million labelled invoices. The system could prevent a food crisis in oil manufacturing. Both Random forest and decision tree methods were used in this approach.

4.4 Previous Work (IA.SAE)

This dissertation will use the real-world example of ASAE operation scenario to implement, test and compare the proposed methods to modify the vehicles' routes in real time. ASAE inspections can be formulated as a DVRP, as unpredictable disruptive events will influence the routes. There are several depots from where the vehicles are dispatched, and these vehicles are assigned routes to visit multiple economic operators during the workday. Work previously made by Telmo Barros in the context of IA.SAE will be used as a starting point for the proposed approach [27] [6] [5]. This dissertation will also include the constraints involved in ASAE operations scenario.

4.4.1 Problem Description

Previous work on the context of IA.SAE Project implemented the route calculation module using several artificial intelligence algorithms, such as exact methods like Branch and Bound and meta-heuristics like Genetic algorithms, Hill climbing, and Simulated Annealing [27]. The work formulated the problem as a Multi-depot Vehicle Routing Problem with Time Windows (MDVRPTW). Several other restrictions were added to the inherent on this type of problem, to respect the constraints on ASAE inspections. The calculated routes are not revised in real-time or changed once a disruptive event occurs; the problem was simplified to a static state. As the ASAE operations environment has numerous factors that might influence these routes, they need to be recalculated

and revised in real-time to achieve further optimality. Not revising the routes in real-time, might also mean that the solution previously calculated can become unfeasible, as the initial resources and assumptions might not remain during the plan's execution.

As previously stated, ASAE comprises three decentralised units subdivided into twelve distinctive operation units. The sixteen units were manually identified and localised geographically in the map using geocoding tools. A fundamental constraint on ASAE context is that each operational unit has a limited territory to conduct the inspections. Different geographical areas were created for each operational unit, thus assigning each economic operator the respective operation unit responsible for the inspection.

4.4.2 System architecture

The system's architecture comprises a MySql database, a web application, and a Rest API for route generation. The developed Rest API performs requests to the Open Source Routing Machine (OSRM) to calculate the routes. OSRM is an open-source routing app developed in C++ and designed to use the OpenStreetMap's data. To calculate the shortest paths and distances, OSRM uses an implementation based on contraction hierarchies and multilevel Djirksa, reducing the computational times compared to other approaches using A* algorithm.[6] The routing app is used to calculate the shortest path between several points in the map, in terms of distance or duration. As a free, open-source platform, OSRM has a limit in the number of requests that can be made to this API. This application can be installed locally, thus removing the request limit.

Each inspection must be done during the working period of the economic operators. Nevertheless, this information is not available in ASAE's system and is described by collaborators as volatile and hard to obtain. For the purpose of the work developed by T. Barros, the schedules were randomly generated using a set of predefined possible ones. Telmo proposed a future implementation to automatically gather each economic operator's schedule using Google Place Details API to tackle this issue. The same inspection can start and end in different days, demanding a two-day representation of the schedule.

4.4.3 Utility Function

The utility of each economic agent is of significant importance in the calculation of the overall inspection route. ASAE was defined as a maximisation problem, aiming to maximise the overall utility of the inspection routes. Each economic operator's utility is calculated based on the past denouncements (complaints) and using a complex function. The function 4.1 approximates a logarithmic function and returns values between 0 and 1. Every economic operator has a minimum utility of 0.05, even in the abscence of complaints, as it might not be compliant with the law. The maximum utility of 1 is given when an economic operator has more than 20 complaints. The

[6] Project Open Source Routing Machine (OSRM), available at http://project-osrm.org/, accessed on 2021-01-31

utility function implementation allows additional parameters that might also influence economic operators' utility.

$$\begin{cases} 0.05 & Complaints = 0 \\ \frac{Complaints}{10} & 0 < Complaints < 10 \\ \frac{Complaints-10}{100} + 0.9 & 10 \leq Complaints < 20 \\ 20 & Complaints \geq 20 \end{cases} \tag{4.1}$$

4.4.4 Algorithms

The storage and representation of the calculated routes are identical between all algorithms. As each algorithm iterates and further optimises, the current calculated routes and economic operators to inspect are saved in memory. Finally, the set of routes outputted by the algorithms in the last iteration needs to be manually accepted, being saved on a database. To calculate the optimised set of routes, the user must select the algorithm to be used. For comparison purposes, the set of algorithms used in this work have the same input parameters. The parameters used were: *unidade*, to specify the corresponding operational unit; *entidades*, a list representing each economic operator (ID, name, schedule, location, utility, etc.); *durations*, a matrix with all the distances from the operational unit and each of the economic operators, *n_fleet*, the number of brigades to be taken into account in the solution, *max_durations*, the maximum work time of each brigade, in seconds; *start_secs*, the time each brigade will start working; *objective_f*, either 0 or 1 to maximise the total sum of utilities of the selected economic operators or maximise the number of economic operators to visit respectively. Several other parameters have been specified, but those are of less importance as they refer to parameters used to tune the algorithms.

The use of Branch and Bound's exact method was discarded and concluded to be of high time complexity when used in VRPs. Factors like the number of brigades, number of vehicles, and economic operators influence this algorithm's computational time, making it unfeasible to use in a reasonable time. Hill Climbing was the most promising method regarding the meta-heuristics, delivering the best utility solution with less computation time than the others.

4.5 Summary

The ASAE case study will be important to this work's development as it will serve as an example of a use case for the proposed approach, allowing it to be field-tested. Using real data from the ASAE's operation environment, the approach will be tested in fundamental factors, like solution optimality and the time taken to obtain a solution. This case study will also allow to empirically verify the benefits of using algorithms capable of revising and modifying vehicle routes in real-time.

Previous work on project IA.SAE statically tackled the problem, calculating the routes before their execution, using a set of optimization algorithms. Their conclusions and approach can be

used to calculate a set of optimized routes previously to an inspection day. The vehicle fleet then executes this provisory plan, and the system revises and modifies it in real-time once disruptions occur.

Chapter 5

Problem and Proposed Solution

This chapter provides an overall view of the problem, taking into account the concepts gathered from studying state of the art. A general formulation to the problem is proposed, as well as specific constraints specific to ASAE context. A solution methodology is also proposed, suggesting both constraints that need to be taken into account upon the route calculating and algorithms that can be used to obtain a solution.

5.1 Problem Description

Although this dissertation is involved with CIGESCOP project and will have the ASAE scenario as a case study, the work will go beyond that. The proposed methodology is expected to be of possible use in other similar scenarios. Essentially, the problem consists of calculating routes for a vehicle fleet that take into account the existence of dynamic environment factors that will influence the vehicles' routes. The operations plan needs to be continuously updated during the planning horizon and revised in real-time, with the help of systems capable of transmitting data in real-time. Regarding the current pandemic context, vehicle routing problems gathered special attention in the last year since they have an essential role in many companies that now work mostly remotely, delivering their packages to the customers instead of selling them on physical stores. Companies need to maximise their profits by adopting an optimised set of routes that can serve their customers at acceptable times. The routes can be calculated and optimised using information available from sources such as road maps, expected travelling time between two rode nodes, expected service times, and vehicle capacity. Although this information allows for calculating optimised routes, it makes the wrong assumption that one route will maintain its optimality along with the plan's execution. Dynamic factors will influence the optimality of the routes, but might also make the set of routes unfeasible; some clients' demands will be unsatisfied. The extra profit of achieving the optimality of the routes and avoiding the costs of an unfeasible route plan are two big motivations to solve the dynamic variant of VRP.

5.2 Problem Formulation

The problem is formulated as a Multi-depot Dynamic Vehicle Routing Problem with Time Windows (MDDVRPTW). This formulation describes the vast majority of vehicle routing problems in the real world. From delivery companies to institutions that perform inspections such as PSP, ASAE and DGAV in Portugal, they usually have several depots from where the vehicles are routed. Time windows are essential, as clients may have specific times to receive a delivery, or in the case of inspection, economic operators' schedules must be respected. The dynamic factors in the real-world environment that might influence the routes cannot be modulated in a single problem, as they are almost infinite.

For this work's purpose, six disruptive elements will be considered: dynamic inspection times, dynamic travel times between two sites, vehicle breakdowns, inspection breakdowns, utility changes and emergency inspections. Inspection times pertain to the time taken for a brigade to inspect one economic operator thoroughly, and this time is continuously variable and cannot be precisely predicted. The vehicle travel time is the time vehicles take to navigate between two nodes in the road network. This time might be influenced by factors such as traffic intensity, closed links in the network (closed roads) or other factors that can condition the vehicles speed. Vehicle breakdown will modify the constraints of the problem as the broken-down vehicle can no longer be used. Inspection breakdowns are disruptions that can happen during the service time, during the inspection in ASAE's context, and will invalidate a particular brigade from continuing its work. Utility changes are changes in the value of each economic operator in the system, meaning it must re-optimise the whole schedule to find the new combination of routes with the best utility. Emergency inspections are unpredicted and known in the runtime and must be performed until the end of the workday. A solution that doesn't inspect the economic operators considered an emergency is an unfeasible solution. All these disruptive events will condition the optimality of the routes and demand them to be revised and recalculated.

The problem addressed in this dissertation is a maximization problem, aiming to find the best feasible solution according to the defined utility function (refer to Section 7.7). In the real-life scenario, there are numerous constraints with higher and lower importance, some that can't even be perceived or translated to the system. This dissertation modulated the problem with a finite number of constraints perceived from the real-life scenario and validated by ASAE collaborators. The following are the most relevant ones:

- The inspection brigades have a defined time to leave the initial depot (operational unit) and arrive at the same place when the workday ends and at a specified timestamp.
- One inspection can only occur if the economic operator is opened (its schedule is available) when the inspection will start.
- The same economic operator can only be inspected a single time during the whole operational plan (among all the different inspection routes).

- There must be an available path between all the economic operators in one inspection route when respecting its sequence.
- Each inspection brigade can only inspect one economic operator at a time.
- The next inspection in the route sequence can only start when the brigade gets to its location.
- A vehicle or brigade that has suffered a breakdown is no longer available for inspection tasks in that workday.
- A brigade must finish an inspection if it starts and no disruption happens.

5.3 Utility Function

The utility function responsible for calculating each customers' utility must be dynamic and allow for changes depending on the context or selections made by a business expert. A customer/economic operator's utility is seen as the company's utility gain when visiting and serving it. This utility can be monetary profit or anything that maximises and contributes to the company reason of existence. Customer satisfaction also plays a fundamental role in the utility function of specific business models, for instance, in newspaper delivery services. When a newspaper is not delivered, the customer complains to the help call-centres and may cancel the subscription, both incurring in monetary losses [38]. The function needs to allow a customer utility bias, as some recurrent customers need to be given higher priority as they bring more value to the company and need to be kept satisfied. Even if a particular order of those customers is of lower utility (for instance, a small demand order), the utility function must consider a "premium" customer and increase the utility of that order accordingly. Contextualising with the ASAE scenario, several macro elements might influence each economic operators' utility. These factors can be regional, enclosed in a specific business area or a combination of both. For instance, one district might have a severe problem in food safety on the butcheries. All the butcheries in that district need to be given a higher utility regardless of other factors in the utility function. Contributing to ASAE's primary goal of increasing food safety in Portugal, economic operators with a more significant number of complaints must be given a higher utility. By inspecting them, ASAE will tend to find more non-compliances with the law and take measures to increase food safety.

This dissertation proposes a complex utility function based on three domains that can be weighted by a business expert, changing the influence of each component in one solution's final utility. The complex utility function aims to maximise the total utility gathered by visiting all the economic operators composing the operational plan, to increase the solution similarity (between the initial solution and the one obtained after addressing disruptions), and decrease the average arrival time to the depot at the end of a workday.

5.4 Geo-referenced information

All the vehicles on the fleet must have their localisation regularly monitored and communicate it in real-time to the control centre. Every economic operator considered in the system must also be geo-referenced in the system's map. With all the entities geo-referenced, the system can adequately track each vehicle route and detect disruptions, both on travel times and service times, but also in case of breakdown. The system will measure the vehicle's operations' delays and compared those to the current execution plan. If the delay is above a defined threshold, the set of routes needs to be revised and optimised. The geo-reference of the entities will be managed using an approach similar to the one used by Barros in IA.SAE [27], based on OSRM. This open-source tool will be responsible for determining the optimal path between two positions on the map.

5.5 Proposed solution

The solution in the context of DVRP must be obtained in a reasonable time, as they are taken in real-time. The revision of the plan must be performed and communicated to the currently operating vehicles in close to real-time. Two time windows need to be taken into account for the solution: the time available for the algorithms to optimise and output a revised solution and the time window necessary to communicate the solution to the vehicles. This last time window, respecting the communication methods, is outside the scope of this dissertation and will be considered instant for simplicity purposes. As the vehicles are in constant operation during the day, the time taken to calculate the problem's dynamic solution will reduce the profits or incur extra losses. For this purpose, exact methods will be discarded as they fail to give an acceptable solution in reasonable times with an increasing number of nodes and vehicles. The solution will contemplate meta-heuristic methods that can obtain a reasonably decent solution with shorter computational times. These methods also benefit from several tunable parameters, such as the number of iterations and the number of individuals in a population (for Genetic Algorithms), that can be adjusted to regulate the trade-off between the solutions optimality and the computational time needed to obtain it. For this dissertation, four optimization algorithms will be implemented and studied: Hill-Climbing, Simulated Annealing, Tabu-Search and Large Neighborhood search.

The solution will be based on a continuous optimisation methodology, storing the current solution to the problem in memory and modifying it periodically (Figure 5.1). The initial solution will be calculated using static VRP methodologies, providing a more accurate base solution with the information available at that time. The algorithms implemented by T. Barros [27] will serve this purpose.

The concept of solution during this dissertation will be the same as the operational plan. An operational plan is a set of inspection routes that are scheduled for execution in a specific workday. Each brigade in the system will be in charge of one inspection route, and all the economic operators that compose it will be inspected if no disruption happens. To simulate a system that works and addresses disruptions in real-time, a disruption Generator module was created. This module

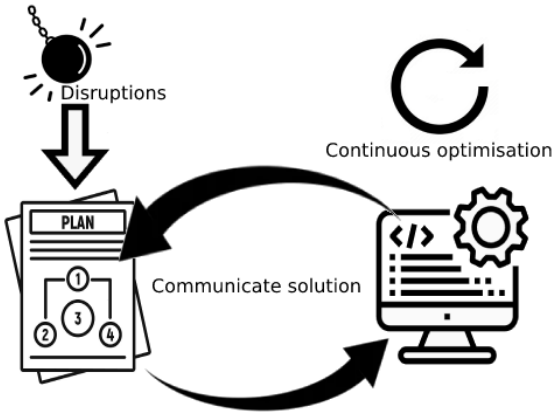


Figure 5.1: Proposed solution scheme

will randomly generate disruptions based on certain parameters and applies them to the inspection routes, updating the problem restrictions and needed variables. To simulate the real-time, a timestamp was created that can determine the current simulation time. The current simulation time indicates which action in the operational plan each of the brigades is. A very simplistic web application was developed to allow the user to control several input parameters and see helpful information such as schedules and maps.

5.6 Performance Evaluation

The approach and the proposed algorithms will be tested and compared using several performance metrics. Several heuristic-based algorithms will be compared using the trade-off between the time taken to obtain the solution and its optimality. The optimality of a solution is directly proportional to the total amount of utility generated by satisfying the operations plan (accomplish all the routes, inspecting all the economic operators). Since this problem concerns a DVRP instance, the similarity between the initial solution and the one obtained after addressing disruptions will be considered. Inspection routes that keep similarities to the initial solution will generate a solution of higher utility. The solution optimality is also dependent on the resources used to accomplish the calculated routes. These costs include the costs of the vehicle fleet operation (fuel and vehicle wear), costs of each brigade's crew, distance travelled by the total of vehicles and total inspections time. For the purpose of this work, these costs will not be taken into account.

Chapter 6

Disruption Generator

6.1 Introduction

Disruptions occur in real-time and influence one or more inspection routes, meaning they can become unfeasible, or there might be room for further improvement of their utility. This work proposes and implements a system that tackles disruptions in real-time, but for the purpose of this work, the "real-time" had to be simulated to allow testing the proposed approach. The Disruption Generator was the solution found to artificially generate disruptions that could be addressed by the algorithms the same way as they would be in a real-life scenario.

The disruption Generator composes an essential module of this work, responsible for generating disruptions to inspection routes following several disruptions templates fine-tuned by a set of parameters. It receives an operational plan corresponding to the set of inspection routes for one working day and returns a new operational plan with disruptions in the inspection routes.

6.2 Disruption Types

This dissertation used feedback gathered from ASAE's inspectors, developing a disruption generator that tries to mimic the conditions and disruptions occurring in a real-life scenario as closely as possible. Six major disruptions were identified: Disruption of the inspection times, disruption of travel times, vehicle breakdowns, inspection breakdowns, utility changes and emergency inspections.

6.2.1 Inspection Time Disruption

Asae inspects economic operators from several business areas that can be classified into ten main clusters. Although one can reasonably predict the inspection time necessary to inspect an economic operator from a specific cluster, many factors can influence these times. The most obvious one is the presence of non-compliance with the law, which may require the application of fines, detentions and shut down the operation. This, allied with other factors, can delay the inspection and therefore threaten the inspection route's feasibility. The system uses a Gauss distribution to

sample a new inspection time, simulating a disruption of the predicted inspection time. The two parameters to calculate the gauss curve are the predicted inspection time, assigned to the distribution's mean, and a deviation factor that the user can customise. The deviation factor adjusts the severity of the disruption, where higher deviation values lead to higher differences between the original and the new inspection times. The disruption severity increases proportionally with the inspection time, meaning that longer inspections will tend to generate a more significant time difference. The code used to generate the new inspection times can be found in listing 6.1.

6.2.2 Travel Time Disruption

Travel time disruptions regard the disruptions caused by varying travel times between economic operators and between them and the depot. These variations can be caused by a panoply of external elements, such as road conditions, traffic conditions or simple changes in the vehicle's velocity, always impacting the estimated travel times. Travel time disruptions are usually associated with delays, where a certain brigade took more time than expected to reach its next destination, an economic operator or the final depot. Similarly to the inspection time disruptions, these are calculated using a Gauss distribution where the mean is the travel time returned by the OSRM server, and the deviation is a parameter specified by the user. The disruption severity increases proportionally with the travel time, meaning that longer travel times will tend to generate a more significant time difference. The code used to generate the new travel times can be found in listing 6.1.

```
1     first_travel = True
2     for i, step in enumerate (route_solution):
3         if step[0] == 'Travel':
4             if travel_time_bol and random.randint(1, 100) <= travel_time_prob: #
                  Modifies the travel time
5                 travel_time = step[1][1]-step[1][0]
6                 new_travel_time = float(getGaussNumber(travel_time, travel_time/
                      travel_time_gauss))
7                 if not first_travel:
8                     step[1] = [route_solution[i-1][1][1], route_solution[i-1][1][1]
                          + new_travel_time]
9                 else:
10                    step[1] = [step[1][0], step[1][0] + new_travel_time]
11                    first_travel = False
12            elif i != 1:
13                travel_duration = step[1][1] - step[1][0]
14                step[1] = [route_solution[i-1][1][1], route_solution[i-1][1][1] +
                      travel_duration]
15        elif step[0] == 'Inspection':
16            if inspection_time_bol and random.randint(1, 100) <=
                  inspection_time_prob: #Modifies the inspection time
17                inspection_time = step[1][1]-step[1][0]
18                new_inspection_time = float(getGaussNumber(inspection_time,
                      inspection_time/inspection_time_gauss))
```

```
19                step[1] = [route_solution[i-1][1][1], route_solution[i-1][1][1] +
                      new_inspection_time]
20            else:
21                inspection_duration = step[1][1] - step[1][0]
22                step[1] = [route_solution[i-1][1][1], route_solution[i-1][1][1] +
                      inspection_duration]
23        elif step[0] == 'Wait':
24            final_wait_time = step[1][1] #wait time
25            if route_solution[i-1][1][1] >= final_wait_time:
26                del(step) #doesnt need to further wait
27            else:
28                step[1] = [route_solution[i-1][1][1], final_wait_time]
29        elif step[0] == 'Depot' and i == len(route_solution) - 1:
30            step[1] = [route_solution[i-1][1][1]]
```

Listing 6.1: Method used to generate stochastic inspection and travel times.

6.2.3 Vehicle Breakdown

This disruption type is common in most VRPs and concerns the fleet vehicles in operation. Any vehicle in operation can suffer a breakdown, caused by various reasons and prevents that vehicle's brigade from proceeding the corresponding inspection route, meaning that that same brigade won't visit the economic operators that were not inspected. In some problems that involve customers, upon a vehicle breakdown, the remaining customers still need to be served, so they are usually distributed by the other operating vehicles in the most optimised way possible. As ASAE doesn't have customers of this type, meaning they don't prepare their inspections (any brigade can inspect any economic operator), the system can simply re-optimise the whole operation schedule searching for the best solution. Since there is one less vehicle available, the system must update its restrictions and delete, from the solution, the route corresponding to that vehicle. The operators in the broken-down vehicle's inspection route might be inspected by other brigades or forgotten if the system doesn't include them in the best solution found. Vehicle breakdowns can only happen when the vehicle is in circulation (from destination to destination), and all the inspections completed by that vehicle's brigade are not influenced. The code used to generate a vehicle breakdown disruption can be found in listing 6.2.

```
1     vehicle_breakdown = random.randint(1, len(solution[1:-1])) #selects a random
          travel action
2     vbreakdown = False
3     last_operator_visited = -1
4     if vehicle_breakdown_bol and random.randint(1, 100) <= vehicle_breakdown_prob:
5         disruptions_generated.append('Vehicle Breakdown')
6         i = 0
7         for j, step in enumerate(route_solution):
8             if step[0] == 'Inspection':
```

```
9                 if vbreakdown:
10                    route_solution[j] = ['Fail_Inspection', 'Cant visit', step[2],
                          step[3]]
11                else:
12                    last_operator_visited = step[3]['id']
13            if step[0] == 'Wait':
14                if vbreakdown:
15                    route_solution[j] = ['Wait', ['-','-']]
16            if step[0] == 'Depot':
17                if vbreakdown:
18                    route_solution[j] = ['Depot', ['-','-']]
19            if step[0] == 'Travel':
20                i += 1
21                if vbreakdown:
22                    route_solution[j] = ['Travel', ['-','-']]
23                elif i == vehicle_breakdown:
24                    vbreakdown = True
25                    breakdown_time = random.randint(int(step[1][0]), int(step
                          [1][1]))
26                    step[1] = [step[1][0], breakdown_time]
27        log = {'type': 'Vehicle Breakdown', 'operator': last_operator_visited}
```

Listing 6.2: Method used to generate the vehicle breakdown.

6.2.4 Inspection Breakdown

This disruption is very particular to ASAE's operations scenario and regards problems and unforeseen events during inspections. Contrarily to the vehicle breakdown, inspection breakdowns can only happen while a particular brigade is inspecting an economic operator. Inspection breakdowns happen when one brigade is forced to stop inspecting an economic operator for some reason or catastrophic failure and cannot proceed with its inspection route. Similarly to Vehicle Breakdowns, the unvisited economic operators don't necessarily need to be inspected and may or may not be distributed by the other operating brigades in the system, searching for the most optimised solution.The code used to generate an inspection breakdown disruption can be found in listing 6.3.

```
1     if inspection_breakdown and random.randint(1, 100) <= inspection_breakdown_prob
          :
2         disruptions_generated.append('Inspection Breakdown')
3         i = 0
4         for j, step in enumerate(route_solution):
5             if step[0] == 'Depot':
6                 if ibreakdown:
7                     route_solution[j] = ['Depot', ['-','-']]
8             if step[0] == 'Travel':
9                 if ibreakdown:
10                    route_solution[j] = ['Travel', ['-','-']]
```

```
            if step[0] == 'Wait':
                if ibreakdown:
                    route_solution[j] = ['Wait', ['-','-']]
            if step[0] == 'Inspection':
                i += 1
                if ibreakdown:
                    route_solution[j] = ['Fail_Inspection', 'Cant visit', step[2],
                        step[3]]
                elif i == breakdown_inspection:
                    ibreakdown = True
                    breakdown_time = random.randint(int(step[1][0]), int(step
                        [1][1]))
                    step[1] = [step[1][0], breakdown_time]
                else:
                    last_operator_inspected = step[3]['id']
        log = {'type': 'Inspection Breakdown', 'operator': last_operator_inspected}
```

Listing 6.3: Method used to generate an inspection breakdown.

6.2.5 Utility Changes

Each economic operator has an associated fixed utility that is retrieved from the database and used in the utility function to calculate the most optimal set of routes. In the real-world scenario, the utility values of the economic operators can change due to micro and macro factors. For instance, discovering several butcheries that sell meat that does not follow the required safety parameters might indicate a localised problem. Therefore, the butcheries' utility in that area needs to be updated because they seem to be a greater chance to be non-compliant with the law. This work uses ASAE's ten cluster divisions of several economic activities to simulate such behaviour: the user inputs an activity code, and all the economic operators belonging to that code get their utility increases by a certain value. The code used to generate new utility values can be found in listing 6.4.

```
    if utility_changes_bol and random.randint(1, 100) <= utility_changes_prob:
        random_activity = random.choice(['I', 'II', 'III', 'IV', 'V', 'VI', 'VII',
            'VIII', 'IX', 'X'])
        schedule = []
        for operator in operators:
            if operator['CODIGOS_ACTIVIDADE'] != None:
                code_array = operator['CODIGOS_ACTIVIDADE'].split(',')
                codes = []
                for code in code_array:
                    codes.append(code.split('.')[0])
                if random_activity in code_array:
                    operator['utility'] += random.random(1,2) #adds a random
                        utility value
```

Listing 6.4: Method used to generate utility changes in specific classes of economic operators.

6.2.6 Emergency Inspection

An emergency inspection is one inspection that is of top priority and must be performed, or the solution will be unfeasible. These types of inspections appear when the operations plan is already in execution and are assigned with a very high utility compared to the maximum of 1 unity for all the other economic operators. The problem restrictions are also updated, and it becomes mandatory to perform all the emergency inspections. To simulate an emergency inspection, the disruption generator selects randomly one economic operator from all the existent ones not present in the initial operational plan. The chosen economic operator is labelled as emergency inspection and appended in a random place in an inspection route. However, emergency inspections can be performed by any brigade, and any brigade is able to perform multiple emergency inspections; what matters is that the whole operational plan performs all of them. The new economic operator is selected from the same operational unit as the inspection route. Therefore, it is usually close to the brigade's area, making it easier to schedule its inspection. This disruption is the one with the most complex implementation among the several disruption types. Adding a new economic operator to an existing inspection route involves recalculating all the timestamps from every route action and calculating the new paths between the new operator and the one before and after.The code used to generate an emergency inspection can be found in listing 6.5.

```
1     operator_to_add = []
2     operator_to_add_index = 0
3     if emergency_inspection_bol and random.randint(1, 100) <=
          emergency_inspection_prob:
4         disruptions_generated.append('Emergency Inspection')
5         economic_operators = getEconomicOperators('UO3')  #MUDAR ESTA HARDCODED
6         random_operator = solution[0]['id']
7         while random_operator in visited_operators:
8             random_operator = random.choice(economic_operators)
9         random_operator['utility'] = EMERGENCY_INSPECTION_VALUE
10        operator_to_add = random_operator
11        log = {'type': 'Emergency Inspection', 'operator': random_operator['id']}
12        #coordinates = getGeoCordinates(random_operator)
13        emergency_inpection_index = random.randint(1, len(solution[1:-1]) + 1)
14        operator_to_add_index = emergency_inpection_index
15        index = 0
16        for j, step in enumerate(route_solution):
17            if step[0] == 'Travel':
18                index += 1
19                if index == emergency_inpection_index:
20                    operator_before = solution[index-1]
```

```
21                      operator_after = solution[index]
22                      travel_time_before = getTravelTime({'lat':operator_before['lat'
                            ], 'lng':operator_before['lng']}, {'lat':random_operator['
                            lat'], 'lng':random_operator['lng']}) #gets the travel time
23                      travel_time_after = getTravelTime({'lat':random_operator['lat'
                            ], 'lng':random_operator['lng']}, {'lat':operator_after['
                            lat'], 'lng':operator_after['lng']}) #gets the travel time
24                      if route_solution[j-1][0] == 'Depot':
25                          list_before = ['Travel', [route_solution[j-1][1][0],
                                route_solution[j-1][1][0] + travel_time_before]]
26                      else:
27                          list_before = ['Travel', [route_solution[j-1][1][1],
                                route_solution[j-1][1][1] + travel_time_before]]
28                      current_inspection = ['Inspection', [list_before[1][1],
                            list_before[1][1] + INSPECTION_TIME], {'utility':
                            random_operator['utility']}, {'id': random_operator['id']}]
29                      list_after = ['Travel', [current_inspection[1][1],
                            current_inspection[1][1]+travel_time_after]]
30                      del(route_solution[j])
31                      route_solution.insert(j, list_after)
32                      route_solution.insert(j, current_inspection)
33                      route_solution.insert(j, list_before)
34                      break
```

Listing 6.5: Method used to generate a new emergency inspection

6.3 Summary

The disruption generator module was developed to artificially generate disruptions on inspection routes, simulating the dynamic behaviour of the real-life scenario. Consulting ASAE's chief inspectors, this dissertation implemented six different disruption types: travel times disruptions, inspection time disruptions, vehicle breakdowns, inspection breakdowns, utility changes, and emergency inspections. These disruptions are generated based on specifiable parameters and are applied to the inspection routes with a defined probability. After the disruption generation, the new operational plan can become unfeasible, being impossible to execute if no changes are made.

Chapter 7

Implementation

7.1 Introduction

This dissertation proposes an approach and implements a system to address operational-plan disruptions, modifying the inspection routes in real-time, maintaining optimality. The solution was developed mainly in Python, as it is an object-oriented, high-level programming language with a syntax that prioritizes readability, therefore reducing the maintenance costs. Python also supports a panoply of modules and packages that can be helpful and used to solve specific tasks in a system. Python is also a powerful language to work with optimization algorithms and programs that involve a large amount of data and in different forms. The approach used in this dissertation is composed of three main modules: the Disruption Generator responsible for generating artificial disruptions, the module responsible for solving the optimization problem, and a middle module responsible for connecting the two previous ones.

7.2 Routing API - Project OSRM

Bearing in mind the conclusions by Barros [27] in the topic of routing and map visualisation, the the Project OSRM [31] was used as the API to calculate the distances, travel times and optimal path between two locations in the system. Each economic operator is georeferenced in the system by its geographic coordinates, latitude and longitude, which are used as an input to the OSRM API. The system developed in this dissertation uses an OSRM image running in a local docker container with the Portuguese map provided by the Open Street Map [1]. This dissertation uses the OSRM "table service" to obtain the travel times between all the available economic operators in the system. This method receives a polyline and returns a matrix containing the travel times between the combination of every location, as shown in Image 7.1. The polyline is built with coordinates pairs of a set of sites. A problem instance with N economic operators will generate N2 entries in the table; the table's main diagonal is always 0, as the distance between a place and itself is 0. The path between two locations is irrelevant; since the algorithms don't consider the roads but one single step between two locations, the travel times being the only information

	A	B	C
A	0	A → B	A → C
B	B → A	0	B → C
C	C → A	C → B	0

Figure 7.1: Distance matrix resulting from 3 different economic operators A, B, and C

needed to solve the optimisation problem. Later, and for visualisation purposes, more requests are made to the OSRM API to get the path between every economic operator composing the solution. This approach saves much computational time, as the path is only calculated for the economic operators that were used in the solution. Generating the path between two locations can be costly, with increasing complexity depending on the distance in-between.

7.3 Map Visualisation

This work used the Leaflet [2] library to create a user-friendly map that could display the OSRM API's outputs. The leaflet is the leading open-source lightweight JavaScript library for interactive maps, with a vast amount of mapping features. This library was used in the web application to display a certain solution, drawing the inspection routes of the whole operational plan. The Marker plugin and some customisations were used to signalise significant landmarks, such as the economic operators and the depots. The map provided by this library is interactive, meaning the user can freely move the camera throughout all the world maps and zoom on interesting areas. Figure 7.2 is an example of a map displaying an operational plan composed of five brigades.

7.4 Web Application

Within the scope of this dissertation, a web application was developed to both visualise information and allow the user to interact with the developed application. The web application was developed using Python combined with the Flask [45], a python's library oriented to web development. The application contains four separate pages, and its development was focused on the functionality aspects, allowing the user to view and interact with important information and metrics from the backend.

The first page is a simple page that lists all the available brigades in the system's database. This page allows the user to choose which routes will compose the operational plane that will be

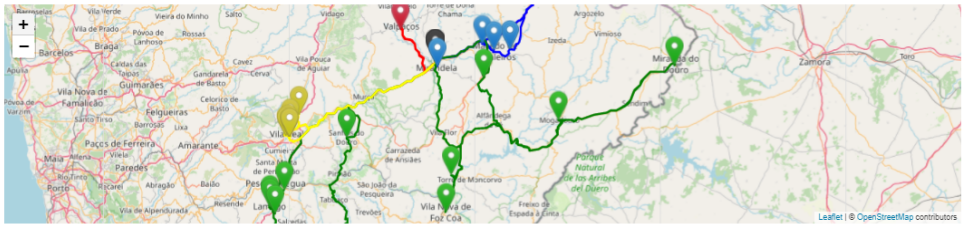


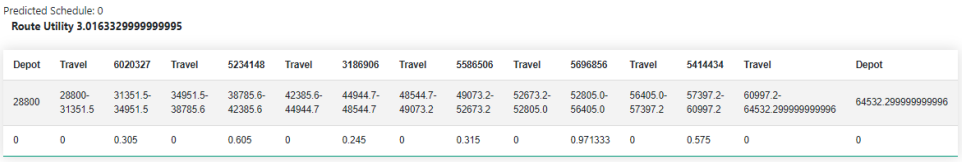
Figure 7.2: Map visualisation of a solution with 5 brigades

solved dynamically by the proposed approach. This user selection will dictate how many brigades will be available in the system and their initial schedule, which can later suffer disruptions.

After selecting the brigades for the operational plan, a page displays a table with the schedule for each considered brigade. The table shows all the economic operators' Ids one brigade is scheduled to inspect, such as the timestamps for the beginning and finish of every operation in the route: leaving the depot, inspections, waiting for an economic operator to open, travelling and arriving at the depot. The timestamps are given in milliseconds starting from midnight of the current day [Figure 7.3]. Likewise, the utility of every economic operator, the total utility of each route and the utility of the complete operational plan are shown on the page. A widget with a map displays the locations of every point of interest. The black marker represents the initial depot, which in this problem is the operational unit. The map also shows the geographic location of each economic operator with the same colour for each of the available inspection routes. The exact path covered by the brigades is identified on the map using a line with the same colour as its economic operators. Lower on the page, the user can specify the parameters of the Disruption generator. It is possible to select the types of disruptions to be generated and the probabilities associated with each one [Figure 7.4]; all the disruptions are generated and applied to the current operational plan.

Following the generation of disruptions, the system recalculates the schedule and respective timestamps for each brigade composing the operational plan. This process is not a re-optimisation, but a simple recalculation of the timestamps, considering the same inspection schedule will be maintained after disruptions. Depending on the severity and type of the disruptions, the schedule can often become unfeasible and therefore not compliant with all the problem constraints. The disruptive schedule is represented in a table, and the inspection routes are on a map, similar to the previous page. Lower on the page is it possible to define the parameters of the optimisation solver. The user needs to select one of the four available optimisation algorithms: Hill-Climbing, Simulated Annealing, Tabu search, and Large neighbourhood search. The three different weights in the utility function are specified and used to calculate the solution's utility during the selected algorithm. Lastly, the current simulation time needs to be filled in milliseconds, indicating the timestamp of the day where the simulation should begin.

The last page concerns the dynamic vehicle routing problem solution. The DVRP instance will be solved by the algorithm and with the settings selected by the user. The system will then output the best solution found by the algorithm addressing all the disruptions generated previously.


Predicted Schedule: 0
Route Utility 3.0163329999999995

Depot	Travel	6020327	Travel	5234148	Travel	3186906	Travel	5586506	Travel	5696856	Travel	5414434	Travel	Depot
28800	28800-31351.5	31351.5-34951.5	34951.5-38785.6	38785.6-42385.6	42385.6-44944.7	44944.7-48544.7	48544.7-49073.2	49073.2-52673.2	52673.2-52805.0	52805.0-56405.0	56405.0-57397.2	57397.2-60997.2	60997.2-64532.299999999996	64532.299999999996
0	0	0.305	0	0.605	0	0.245	0	0.315	0	0.971333	0	0.575	0	0

Figure 7.3: Example schedule for one inspection route

The information is represented similarly to the previous two pages; there is a map containing the representation of all the inspection routes in the operational plan and a schedule table showing all the actions and intermediate timestamps of an inspection route

7.5 Data Structures

7.5.1 Solution representation

From all the data structures used, the most important one was the one adopted to store and represent the solutions for the problem. A solution represents the set of inspection routes to be performed in one working day, and it has to accumulate enough information for the system to compute each of the intermediate timestamps for each action (calculate the predicted schedule). The solution is represented by a list of lists for this work's purpose, taking advantage of python's list capacities and primitives. The outside list represents the operational plan, and the lists inside represent each inspection route. Each inspection route contains all the steps, both economic operators and depots, one brigade has to cover to accomplish the work predicted for one day. Each inspection route can leave the initial depot at a different time interval, starting to work at different times of the day. The time one brigade starts working is defined in milliseconds and stored in a list. The values of each list index correspond to the start time of the inspection route with the same index in the solution. Different economic operators might require distinct times intervals for inspection because of the type of economic activity they belong to, their geographic location, their operation size, and others. The required inspection times are stored in a Python dictionary, where the keys are the economic operator's identifiers, and the values are the expected time intervals to inspect them.

7.5.2 Economic operator / Depot

The economic operators are the places with the potential for inspection, while depots, or operational units, concern the location where the brigades depart and arrive at the start and end of the workday, respectively. Both economic operators and depots are represented using the same data structure and with minor differences. For the purpose of this section, depots will be treated as economic operators. The operators are defined as a python's dictionary, making it easy to access all their information fields. A name and a unique ID identify each economic operator, but only the unique ID is considered in this dissertation. Each economic operator also contains a pair of coordinates (latitude, longitude) with its respective location on the map; these are used to calculate

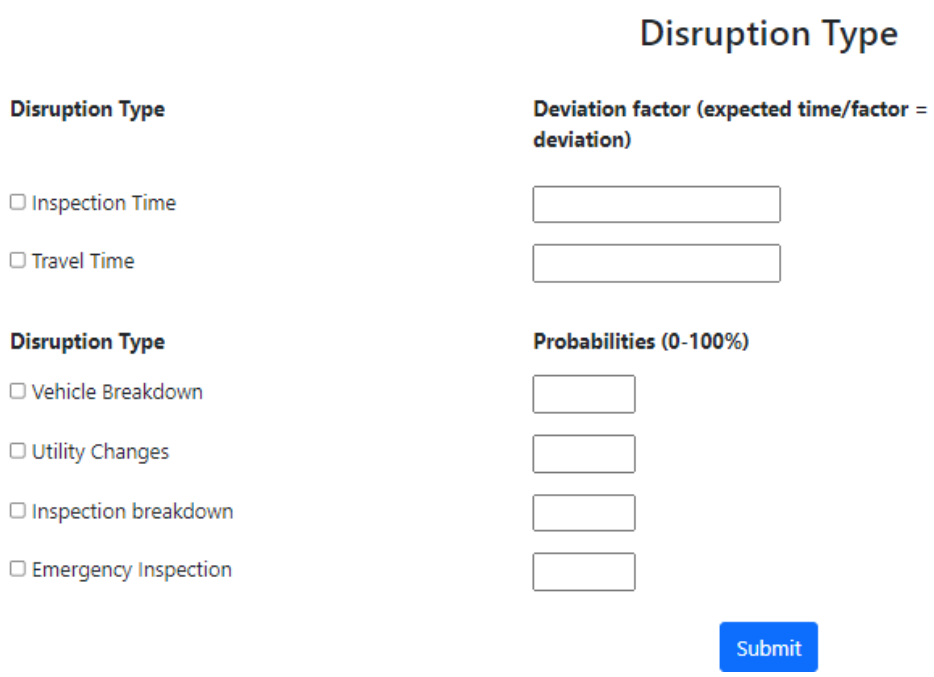


Figure 7.4: Parameters used to specify the disruptions to be generated

the distances between two economic operators. There is an associated utility with every economic operator, represented by a decimal number between 0 and 1 and in the case of depots, this value is 0 (there is no utility gain when visiting a depot). Each economic operator is associated with economic activity, from the ten activities categorised by ASAE and a CAE (activity code). Finally, each economic operator has information about their schedule for all the weekdays, indicating what periods it will be opened and for how long.

7.5.3 Travel Times

Computing the travel times and path between two locations in the system can be time-consuming and impossible to compute during the execution of one algorithm since the utility function run a considerably large amount of times in concise periods. The approach used calculates the travel times previously from the algorithm execution, avoiding awaits during the execution of the utility function. This is possible because the available places on the map, economic operators and depots, don't change in real-time. When the system starts running, it calculates the travel times from every economic operator or operational unit to all others and stores it in a two-dimensional matrix. The developed approach uses OSRM's primitive table service to achieve such a result, as it returns a table with all the distances from a coordinates string (latitude and longitude) representing places in the map received as an argument. The indexes of the rows and columns represent the index in the argument string, and the cells contain the time taken from index x in the row to index y in the column. An indexing list was created, indexing each economic operator's ID with its

corresponding index in the matrix, facilitating its use. Each time the system needs to compute the travel time between two points in the map, it retrieves it from the matrix in polynomial time.

7.6 Schedules

In a real-life scenario, each economic operator has an operating schedule associated that may be different for each day of the week and when compared to other economic operators. The working schedule provides information about what days of the week it will be opened and for what periods of time. Schedules are very important for the inspections as one inspection can only begin if the economic operator is open and the brigade can enter. In most VRPs with working schedules described in the literature, the vehicle has to arrive at a site when it is available and finish its service before it closes. ASAE, as a government institution, doesn't need to respect an economic operator's schedule fully. After an inspection starts, the economic operator must stay open until the brigade finishes the inspection and takes necessary measures, even if it goes against their schedule. The approach implemented allows the user to choose between the two scenarios: a brigade can inspect an economic operator if it started when the operator is open and if the service time doesn't surpass its close time [Listing B.2]; a brigade can inspect an economic operator as long as the inspection starts when the operator is open [Listing B.1].

Algorithm 1 Function used to check a hard schedule. Returns the start time of an inspection or False if the economic operator can't be inspected. Code used available in Listing B.2

for `turn in schedule` **do**
  $temp \leftarrow current\_time + inspection\_time$
  if $(schedule\_start \leq temp) \wedge (schedule\_end \geq temp)$ **then**
    if $(current\_time \geq schedule\_start)$ **then** $Inspection\_start()$ ▷ Inspection can start now
    end if
  else if $current\_time < schedule\_start$ **then**
    $temp \leftarrow schedule\_start + inspection\_time$
    if $temp \leq schedule\_end$ **then** $Inspection\_start()$ ▷ Inspection can start later
    end if
  end if$InspectionFailed()$ ▷ Failed to inspect
end for

Algorithm 2 Function used to check a soft schedule. Returns the start time of an inspection or False if the economic operator can't be inspected. Code used available in Listing B.1

for `turn in schedule` **do**
  if $(current\_time \leq schedule\_start)$ **then** $WaitToOpen()$ ▷ Waits for economic operator to open
  else if $(schedule\_start \leq current\_time) \wedge (schedule\_end \geq current\_time)$ **then** $Inspection\_start()$ ▷ Inspection can start now
  end if
$InspectionFailed()$ ▷ Failed to inspect
end for

Economic operator's Schedules have paramount importance because they are one of the several restrictions in this problem. Nevertheless, this information is not available in ASAE's system and is described by collaborators as "volatile and hard to obtain". [27] For the purpose of the work developed by Barros [27], the schedules were randomly generated using a set of pre-defined possible ones. Barros proposed a future implementation to automatically gather each economic operator's schedule using Google Place Details API to tackle this issue. The same inspection can start and end on different days, demanding a two-day representation of the schedule.

7.6.1 Schedule Representation

In this dissertation, the time unit used was the second, and a schedule is represented by a python dictionary containing all days of a week. Each day of the week includes a list with pairs *(start, finish)* indicating the time in milliseconds when an economic operator will open and close. A specific place can open and close doors multiple times a day, accounting for the lunch and dinner pause and other needed pauses. A day starts at millisecond 0 and finishes at 86400. Figure 7.5 shows the example of a weekly schedule for a specific economic operator.

7.6.2 Schedule Generation

This dissertation, and to generate the economic operator's schedules, implements an approach similar to Barros's but considers the division in ten main areas of activity. Twenty different schedules corresponding to random economic operators in ASAE's database were manually extracted using Google Places. These schedules were then processed and transformed into the implemented schedule data structure; two economic operators were selected from each of the ten types of economic activities. The approach used to generate a schedule for all the database's economic operators was to choose one of the two schedules available for that operator's corresponding type of economic activity. Some economic operators belong to more than two types of economic activities, and in these cases, their schedule is generated randomly from all the template schedules corresponding to the operator's economic activities.

```
'Monday':[[30600,46800],[54000,72000]], 'Tuesday':[[30600,46800],[54000,72000]],
'Wednesday':[[30600,46800],[54000,72000]], 'Thursday':[[30600,46800],[54000,72000]],
'Friday':[[30600,46800],[54000,72000]], 'Saturday':[[30600,46800]], 'Sunday':[]}
```

Figure 7.5: Representation of an economic operator's schedule.

7.7 Utility Function

The utility function used in this work contemplates three different domains. The utility of a particular solution is obtained by weighing the total sum of all economic operators' utilities, the similarity between the initial solution and the solution obtained after solving the disruption, and

the average time each brigade arrives at the depot at the work day's end. This utility function privileges the routes that visit the economic operators with the most utility combined and where the brigades arrive at the depot earlier while keeping the solution close to the original plan, avoiding massive changes. The proposed utility function is flexible, allowing the user to set each of the components' desired weight in the final utility.

7.7.1 Economic Operator's utility

The approach developed by Barros [27] was used to calculate the utility of each economic agent in the system. Each economic operator in the database system has an individual utility value associated that determines how desirable it is to be inspected, the value it brings to ASAE's inspection system. This value ranges from 0 to 1 and is calculated based on each economic operator's past complaints registered on ASAE's system. The attribution of a utility to each economic operator follows the complex function illustrated in Table 7.1, proposed and implemented in the system developed by IA.SAE. No economic operator is assigned with a null value of utility as the absence of past complaints does not guarantee that one establishment is compliant with the law. For this purpose, a minimum value of 0.05 is assigned to all economic operators. The economic operators who have a number of past complaints between 1 and 9 are set with a function that has a more significant slope than the one used to assign economic operators between 10 and 19 past complaints. This function targets the majority of the economic agents in the system (economic operators with less than 20 complaints), accounting for 99.87% of the population of economic operators in the system. Economic operators with more than 20 complaints in the system are assigned a maximum utility value of 1.

Table 7.1: Complex utility function used to calculate a singular economic operator's utility

NPastComplaints	**Utility**
0	0.05
< 10	$\frac{NPastComplaints}{10}$
< 20	$0.9 + \frac{NPastComplaints - 10}{100}$
≥ 20	1.0

7.7.2 Solution similarity

A key element in a dynamic vehicle routing problem is the magnitude of the changes in the routes once disruption occurs compared to the routes initially calculated. These changes entail added costs that may sometimes outcome the savings achieved by re-optimising the routes once disruptions occur; these costs can be hard to quantify. In this problem's context, few domains were

considered to quantify the changes applied to the initial plan. Each route composing a specific solution is performed by a singular vehicle, and the measurement of the changes was encapsulated inside each route, meaning that any similarity between different routes in the same solution is not captured (ex: exchange of operators between two routes is not considered as a similarity in this problem). To output the utility of a new solution, the systems stores the problem's initial solution and compares it to the new solution, calculating a similarity ratio. The average similarity between each corresponding routes in two solutions is used to determine the similarity between two different solutions, meaning that the routes are compared respectively between the initial solution and the solution obtained after addressing the disruption. The similarity between the two routes increases as they encompass the same number of economic operators, and the economic operators visited by the corresponding routes in the initial and posterior solution are equal. Each brigade's order to inspect the several economic operators composing one route influences their similarity. Although not as important as the same number of inspections or similar economic operators to be inspected, each inspection's order also influences the solution's added costs, especially to the human brigades. Brigades might have appointed or scheduled special events, like lunch or other intermediate stops that can become impossible with the new routes. These "field" restrictions are not considered when calculating the solutions, but they cant be disregarded as they occur in real-life scenarios. In this work, "field" restrictions are getting considered because similar solutions that satisfy more the "field" restrictions have a higher utility value. The code used to calculate the similarity between two routes is shown in Listing 7.1.

```
    routeAIDs = []
    routeBIDs = []
    for op in routeA[1:-1]:
        routeAIDs.append(op['id'])
    for op in routeB[1:-1]:
        routeBIDs.append(op['id'])

    if(len(routeAIDs) < len(routeBIDs)):
        temp = routeAIDs
        routeAIDs = routeBIDs
        routeBIDs = temp

    n_operator_equal = 0
    n_order_diff = 0

    for i, op in enumerate(routeAIDs):
        if op in routeBIDs:
            n_order_diff += abs(i-routeBIDs.index(op))
            n_operator_equal += 1
    max_len = max(len(routeAIDs), len(routeBIDs))

    #Calculates the order difference ratio
```

```
23      ratio_1 = (1 - (n_order_diff / (max_len * (max_len - 1)))) * (n_operator_equal
            / max_len)
24      #Calculates the different operators ratio
25      ratio_2 = n_operator_equal / max_len
26      similarity_ratio = ratio_1 * OPERATOR_ORDER_SIMILARITY_RATIO + ratio_2 *
            DIFFERENT_OPERATOR_RATIO
27      return similarity_ratio
```

Listing 7.1: Calculation of two inspection routes similarity

7.7.3 Average arrive time

The last parameter in the utility function is the time each brigade arrives at the depot at the end of the workday. This approach benefits solutions that arrive earlier than solutions that use a greater portion of the available time. The potential utility of a certain solution increases as the brigades arrive later to the depot at the end of the workday. The higher utility of the inspected economic operators shouldn't be seen blindly as a better solution. The solution with the higher sum of the economic operator's utility might entail that the brigades arrive too close to the maximum allowed time. In a real-life scenario, coming too close to the allowed time can arise problems as disruptions in the last steps of the plan's execution can cause one or more brigades to get late. Getting late to the depot means the human workers work above the pre-defined time, which may incur extra crew costs—maximising the total utility gained by inspecting all the economic operators while at the same time, benefiting solutions that arrive earlier may lead to a solution that performs better in a real-life environment.

7.7.4 Unfeasible Solutions

This dissertation work explored unfeasible solutions as they might accelerate discovering new solutions on some algorithms and contributing to a faster convergency to a better solution. Unfeasible solutions are solutions that do not fulfil one or more of the problem's restrictions. Therefore, they are impossible to apply to real-world problems as they will lead to the reach of impossible states. On the other hand, unfeasible solutions allow for a faster cover of the solution space, as they allow to cross the unfeasible regions to reach other feasible regions potentially containing better solutions. The utility of an unfeasible or viable solution is calculated similarly using the complex utility function described in section 7.7. In order for the algorithm to distinguish and weight each solution, unfeasible solutions have to be penalised in the utility function, so they are considered worse. Independently of the solution's utility, the algorithm will always output a feasible solution in case that exists.

7.7.4.1 Penality Types

For the purpose of this work, six types of penalties were taken into account, addressing all the problem's constraints. The penalty is a decimal value ranging from 0 to infinite to represent how

far a solution is from the feasible space. The final penalty value results in the sum of all the penalties resulting from the restrictions some solution is violating. The penalty value also scales proportionally to the severity of the non-fulfilment of specific constraints, meaning the farthest a particular solution is from the feasible space, the more penalty value the system will add to that solution.

Penalty Only_Inspect: One of the constraints respects the time a specific inspection route has to be completed, the time the brigade is allowed to arrive at the operational unit at the end of an inspection route. A penalty is applied to the solution if, ignoring the travel time between each economic agent and the depots, a particular inspection route cannot inspect all the economic operators in reasonable time and respecting their time schedules.

Penalty Impossible_Operators: The number of operators impossible to inspect (their scheduled inspection ends after the allowed timestamp) influences the value of the penalty to be added; more impossible operators mean higher values of penalty. This methodology allows to quickly assess and penalise a solution that will never be possible to accomplish, as after adding the travel times, it will be more unfeasible. To be successfully inspected, the economic operators must be opened on that day of the week. A certain penalty is added for each operator in one inspection route with no available schedule on the corresponding weekday.

Penalty Repeated_Operators: This penalty regards solutions that visit the same operator multiple times: to accomplish a successful inspection, only one of the brigades needs to visit the economic and a single time. The system adds a penalty for each economic operator visited more than once during the whole operation plan. In the case of the disruptive event where an emergency inspection is requested, there is a constraint that this new selected economic operator must be inspected in that operation plan by any composing inspection routes. For each emergency inspection not accomplished, an increasing value of penalty is added, meaning that solutions that fail more emergency stops should be further penalised.

Penalty Path_Error: The OSRM API calculates the several paths and distances between the economic operators and depots, returning everything in a matrix. These individual paths might be impossible if no available path (sequence of roads) connect the two nodes. A route is unfeasible if there is no known path between two consecutive elements of a route (depot or economic operators). The penalty added in this case is proportional to the number of discontinuities in each route composing the operation plan.

Penalty Last_Time: This penalty is probably the most complex one to calculate and respects to the timestamp where the brigades have to reach the depot at the end of the day. This timestamp is customisable and fixed during the algorithm execution, and any brigade that arrives at the depot after that will produce an unfeasible solution. The calculation of this timestamp is very complex and requires much computation. The system needs to iterate through all inspection routes composing a solution, calculating all the times in one solution. Travel times between locations, operational schedule from the economic operators to be inspected and inspection time for each of the operators. After computing all the intermediate steps to execute one solution, the system can infer all the last_time (when the brigade reaches the depot at the end of the day) for each inspection

route. The system applies these penalties to an unfeasible solution, using the sum of times from each route after the last allowed timestamp. This approach will further penalise solutions above the maximum workload limit for the day and play the most important penalty role, as otherwise, a solution would be infinite (as any new operator always adds a positive value of utility).

Penalty Operator_Schedule: This last penalty regards the operator's schedule. Inspecting an operator means its schedule must be available at the time of inspection; the operator has to be open. The system calculates the time required to travel and inspect all the economic operators in the same order as the solution's sequence. It then checks if the operator is available at the time necessary. The system will compute the expected inspection time and proceed to the next inspection when it is available. In the cases where the operator isn't available, there are two possible cases: either the system waits until the operator opens (it generates a solution where the brigade will have to wait for the operator to open, in cases where there will be an available schedule later on) or the solution is unfeasible (there is no free schedule later on that day). The second case entails penalising each operator who couldn't be inspected during the prediction of the whole operational plan. This penalty shouldn't be misunderstood with penalising operators that are not available on a certain day. Being available on a particular day (operating on that day) doesn't mean the operator can be inspected; the operators scheduled to be inspected former can cause impossibilities in the inspections upstream.

7.7.4.2 Penalty techniques

Long [29] studied penalty functions applied to constrained optimisation problems on their work involving particle swarm optimisation. They propose a momentum-type particle swarm optimisation (PSO) method to enhance both the computational efficiency and the solution accuracy compared to the base PSO. This work also suggests a continuous non-stationary penalty function to punish solutions that don't fulfil the problem's constraints. There are two groups of penalty functions: stationary and non-stationary; this work will apply a non-stationary penalty function. Contrarily to the stationary penalty functions, a non-stationary penalty function changes dynamically throughout the search and typically depends on the iteration number. Non-stationary penalty functions were developed and described in the literature as almost always superior in getting the best results when compared to stationary.

This dissertation adopted and implemented the non-stationary penalty techniques studying and implemented by Long. The complex penalty function 7.1 is divided into two functions: a dynamic function $f(x)$[7.2] and a continuous assignment function $H(x)$[7.3] that can address both linear and non-linear constraints.

$$F(x) = f(x) - C(k)H(x) \tag{7.1}$$

The equation 7.1 was adapted to the maximisation problem studied with this dissertation. This equation returns the new utility value after applying the respective penalty values to unfeasible solutions that don't fulfil one or more of the problem's restrictions. The new utility values are

used as normal with the respective solution in the algorithms' execution. The $f(x)$ is the utility value of a specific solution calculated used the utility function described in section 7.7.

$$C(k) = (c \times k)^{\alpha} \tag{7.2}$$

The dynamic function 7.2 depends on the iteration number k and increases as the search progresses. c and α are two problem-dependent constants manually tuned to the values of 0.05 and 1, respectively.

$$H(x) = \sum_{i=1}^{m} [\theta(q_i(x)) \times q_i(x)^{\rho(q_i(x))}] \tag{7.3}$$

The function $H(x)$[7.3] represents the penalty factor. This function will be the sum of all the penalties types applied to a particular solution meaning a solution that violates more constraints is more likely to get a higher value of penalty and therefore a lower utility. The function $q_i(x)$ is a numeric value representing how far a solution is from the feasible space for one penalty type. Function $\rho(q_i(x))$ adjusts the violating function and is set either for the value one when a solution is near the feasible space, or two otherwise.

$$\theta(q_i(x)) = a \times (1 - \frac{1}{\varepsilon^{q_i(x)}}) + b \tag{7.4}$$

The function $\theta(q_i(x))$[7.4] is a continuous assignment function also adapted from Long's work. In this dissertation, a and b are problem-dependent constants that were adjusted for the values of 150 and 1, respectively.

7.8 Solution Generation

The implemented algorithms imply using methods to generate new solutions based on solutions provided as input. For the purpose of this work, six different operators were implemented as shown in Figure 7.6 and 7.7. Depending on the chosen algorithm, these operators are selected and used both in a randomised or sequential way (to balance the way new solutions are generated in a certain iteration). The operators share the common task of taking a solution as input, and output a new solution after changing one or more inspection routes. All the operators used to generate a new solution concern the level of economic operators as these are the blocks that will be moved and changed in place to originate a new different solution. Each operator involves stochastic decisions to select for which route the operation will be applied and which economic operators will be selected. It is important to note that the operators are only responsible for generating a new solution, they don't consider the utility values, and the new solution can be unfeasible. Some of the operators can fail to execute if the solution given as input isn't compliant with some operator's rules. In these cases, and if the operators are being chosen randomly, the system will randomly pick an operator until the chosen one doesn't fail to execute. Otherwise, if the operators

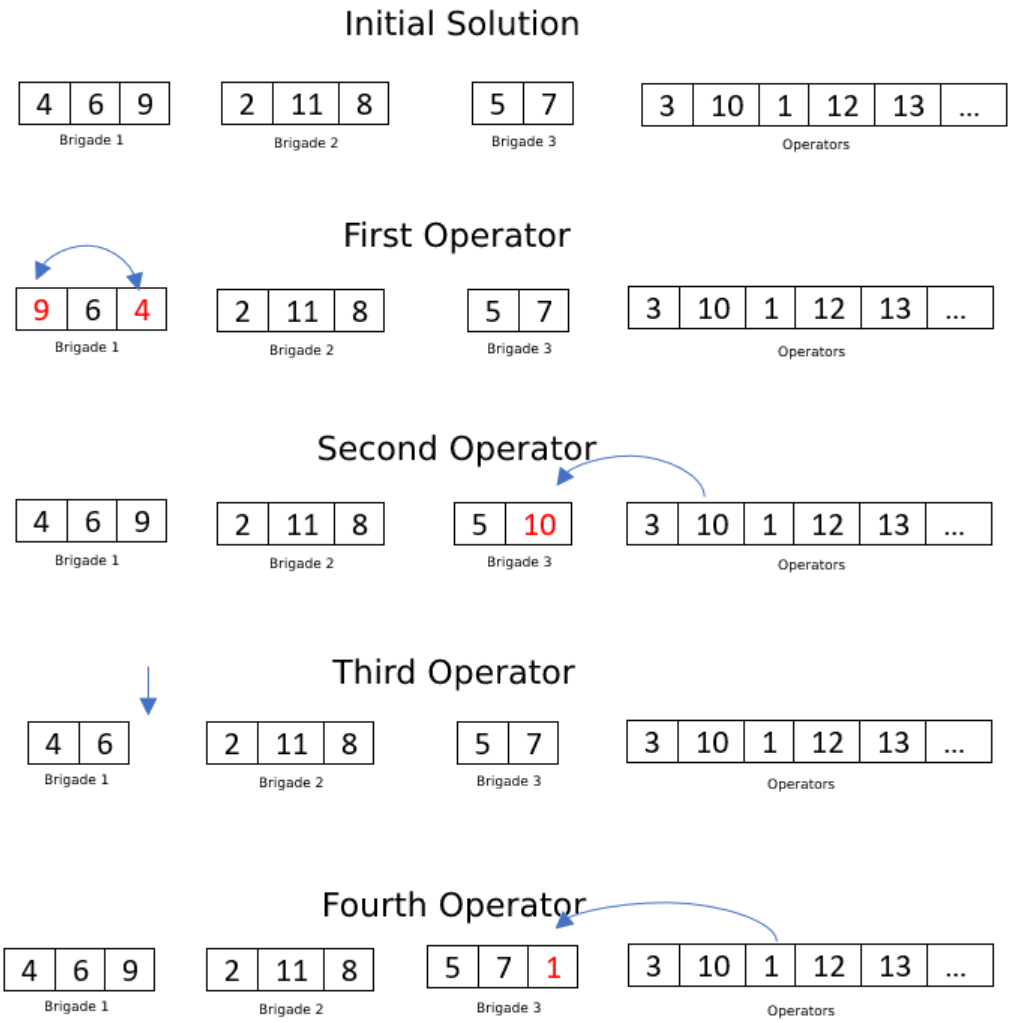


Figure 7.6: Example of different operations on a solution with 3 brigades and 13 economic operators available; The numbers represent the economic operators' ids, and the colour red indicates the changes.

are chosen sequentially, the system will ignore that operator and execute the next one until the selected operator executes without failure.

The first operator (Figure 7.6) consists in exchanging two economic operators in the same inspection route chosen randomly; all the inspection routes maintain the same length.

The second operator (Figure 7.6) changes one economic operator, scheduled to be inspected by one of the brigades, to one in the complete list of economic operators; the inspections routes maintain the same length.

The third operator (Figure 7.6) removes one economic operator selected randomly from an inspection route. This operator only executes successfully if the inspection route contains at least one economic operator scheduled for inspection; the inspection route will decrease one economic operator in length.

The fourth operator (Figure 7.6) adds a new economic operator to one of the inspection routes.

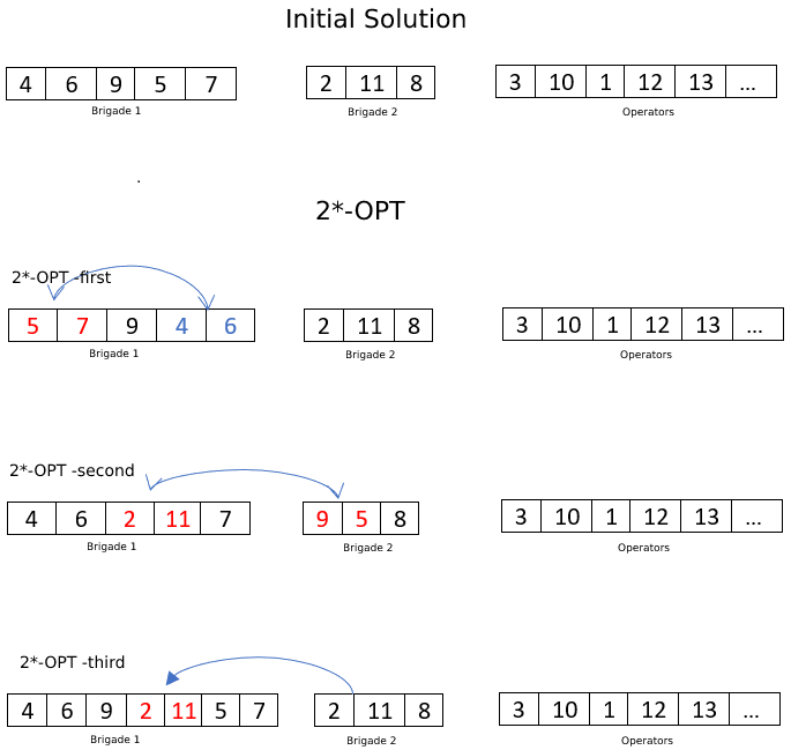


Figure 7.7: Example of different operations using the 2*-OPT on a solution with 2 brigades and 13 economic operators available; The numbers represent the economic operators' ids, and the colour red indicates the changes.

The new operator is chosen from the complete list of economic operators; one of the inspection routes increases its length by one economic operator.

The fifth operator (Figure 7.7) exchanges one economic operator between two inspection routes selected randomly; all the inspection routes keep the same length, and the same list of economic operators will be inspected (by different brigades).

The sixth and last operator, called 2*Opt, is the most complex and can be subdivided into three other operators, one chosen randomly each time the parent operator gets executed. It's called the 2*Opt because instead of operating in blocks of one economic operator, it uses a sublist of two consecutive ones. The 2*Opt implies that the selected inspection routes have at least two economic operators; otherwise, this operator will fail to execute. The first sub operator exchanges two sublists in the same inspection route, maintaining the same number of economic operators. The second sub operator consists in exchanging a sublist from two inspection routes selected randomly. The third operator is similar to the second, but it exchanges the sublists without eliminating them in the original inspection route; the operators will be doubled.

7.8.1 Hill Climbing Algorithm

The Hill-climbing algorithm has the most straightforward implementation and only needs one variable to save the current solution, which is also the best solution found so far; the corresponding utility of that solution is also stored in memory. The first part of the code is common to every algorithm and selects the day of the week a particular solution represents. This day is used to grab each economic operator's schedule when it is needed to calculate its availability for an inspection.

The algorithm starts with a preliminary solution that can be feasible or unfeasible, representing the several inspection routes composing an operational plan. The algorithm calculates its utility and admits that solution as the best one found so far by the search. The algorithm follows the same logic in each following iteration, first generates a new random solution based on the current one, using one of the operators described in section 7.8, and then evaluates its utility using the utility function described in section 7.7. The current solution is updated to the new solution if a null or higher utility improvement was verified. The pseudocode for the algorithm can be found in algorithm section 3.The python code used to implement the algorithm is shown in Listing B.3

Algorithm 3 Hill-Climbing Algorithm

$best\_sol \leftarrow initial\_solution$
$best\_sol\_utility, penality \leftarrow SolutionUtility(best\_sol)$ ▷ Calculate the solution utility
while $i \leq N_{steps}$ **do**
  $new\_sol \leftarrow GenerateNewSolution(best\_solution)$ ▷ Generate New Solution
  $new\_sol\_utility, penality \leftarrow SolutionUtility(new\_sol)$ ▷ Calculate new solution utility
  if $penality > 0$ **then**
    $new\_sol\_utility \leftarrow new\_sol\_utility - PenalityFunction(penality, i)$ ▷ Apply penality
  end if
  if $new\_sol\_utility \geq best\_sol\_utility$ **then** ▷ Updates best solution
    $best\_sol\_utility \leftarrow new\_sol\_utility$
    $best\_sol \leftarrow new\_sol$
  end if
  $i \leftarrow i + 1$
end while

Figure 7.8 illustrates the evolution of the Hill-Climbing search. The yellow line represents the best solution in the current iteration. Unfeasible solutions were removed from the graph to allow a more straightforward understanding of the algorithm. The algorithm generates solutions with different utilities throughout the search, but only the better ones are updated as the current solution.

7.8.2 Simulated Annealing Algorithm

The SA algorithm is an algorithm that allows the acceptance of worse solutions during the search phase, and hopefully, avoid getting trapped in local maximums. This dissertation implements a method that calculates the initial temperature and a cooling schedule inspired by the literature. The algorithm's initial temperature is chosen using a method adapted from Atiqullah [4]. This method

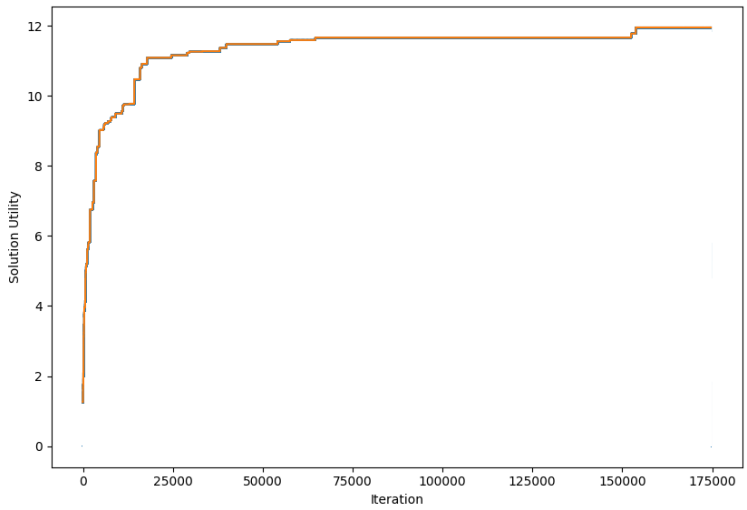


Figure 7.8: Hill Climb solutions utility throughout the search

creates a normal distribution of ten thousand neighbours and calculates the standard deviation to be later used in the equation. The neighbours are generated stochastically using the operators described in section 7.8 over the initial solution. The approach used in this dissertation specified an initial acceptance ratio of 90%, meaning 90% of the neighbours are expected to be accepted at the initial temperature.

```
1     sol_utility, last_time, penality = algorithms_functions.solution_utility(
          inspection_times, solution, solution, indexes, travel_times, starting_times
          , weekday, utility_function, emergency_operators)
2     utilitydiff_data = []
3     cnt = 0
4     for i in range(10000):
5         new_solution, new_sol_utility = algorithms_functions.generateSolutions(
              inspection_times, solution, solution, operators, indexes, travel_times,
               starting_times, weekday, 0, utility_function, emergency_operators)
6         utilitydiff = abs(sol_utility - new_sol_utility)
7         utilitydiff_data.append(utilitydiff)
8         cnt += utilitydiff
9
10    h = sorted(utilitydiff_data)  #sort
11    fit = stats.norm.pdf(h, np.mean(h), np.std(h))
12    acceptance_ratio = 0.9
13    standard_deviation = np.std(h)
14
15    delta_C = cnt/len(utilitydiff_data)
16    temp_0 = (delta_C + 3*standard_deviation)/np.log(1/acceptance_ratio)
17    return temp_0
```

The colling schedule used in this implementation is an adaptation of the parametric cooling schedule used by Atiqullah [4]. In Equations 3.4, the final temperature was set to 0.0001, and the values for the constants a and b were assigned to $a = 2$ and $b = 1/3$. Two new temperature rules were implemented to complement the proposed cooling schedule and based on the two phases of a SA search, global positioning, and solution refinement. In the first iterations of the algorithm, the global positioning phase, the temperature remains at its max for a defined number of iterations. The number of iterations where the temperature remains unchangeable was defined as 5% of the total number of iterations allowed. Maintaining a high temperature for several iterations allows the algorithm to better cover and explore remote areas of the search space without being trapped. The approach developed in this dissertation also adapts the cooling schedule to the solution refinement phase. To better address this search phase, the temperature is set to a null value, meaning the algorithm will behave like an HC algorithm, only accepting the best solutions; it is expected that the algorithm will easily find the local maximum located in the current search area.

The stopping criteria used for the algorithm results from the combination of a defined number of iterations and a defined number of chains without improvement. The algorithm is scheduled and will always execute for a certain number of iterations. After that number of iterations is completed, the algorithm will run indefinitely, while for one hundred Markov chains, there is an improvement above a defined threshold.

Since the SA is a stochastic algorithm, it is not guaranteed always to find the same solution or solutions of the same utility. By repeating the algorithm execution over the same solution, the probabilities of finding a better solution increase, as the steps during the search will be different and lead to a different outcome. This dissertation implements a reheating method that restarts the search process a defined amount of times, with the initial solution and temperature. In each reheat, the best solution found is stored and compared to the best global solution from all the reheats; if the solution from one reheat was better, the best global solution is updated.

The pseudocode for the simulated annealing implementation can be found in algorithm section 4.The python code used to implement the algorithm is shown in Listing B.4

This implementation of the SA algorithm uses a neighbourhood of six solutions instead of the typical one solution neighbourhood. A new python function was created to generate six neighbours from the current solution in the current search iteration [B.5]. This function uses the complete list of operators one single time each to produce at most six new solutions: It is vital to notice that an operator can fail and return no solution; in such cases, the neighbourhood will have a size smaller than six. This function generates the new solutions and calculates their respective utility, returning the one with the highest utility among the neighbourhood. This procedure will help the algorithm analyse several neighbours and increase the chance of selecting a solution that belongs to the feasible zone. The probabilities of all the operators to generate an unfeasible solution is lower than by visiting a single neighbour each iteration.

Figure 7.9 illustrates one execution of the Simulated Annealing algorithm. The blue line represents the current solution at a certain and the yellow line the best solution found. In this graph, it is possible to see the several search phases characteristic of this algorithm. In the first iterations,

Algorithm 4 Simulated Annealing Algorithm

$best\_sol \leftarrow initial\_sol$
$best\_sol\_util \leftarrow initial\_sol\_util)$ ▷ Calculate the solution utility
$temp \leftarrow CalculateInitialTemp(initial\_sol)$ ▷ Calculate the initial temperature
while $reheat < REHEAT\_MAX$ **do**
  $i \leftarrow 0$
  $current\_sol \leftarrow initial\_sol$
  $current\_sol\_util \leftarrow initial\_sol\_util$
  $no\_update \leftarrow 0$
  while $i < MAX\_MARKOV\_CHAIN \wedge no\_update < IMPROV\_TRESHOLD$ **do**
    $new\_sol, new\_sol\_util \leftarrow GenerateNewSol(current\_sol)$ ▷ Generate New Solution
    $util\_difference \leftarrow new\_sol\_util\ curent\_sol\_util$
    if $random() \leq \varepsilon^{\frac{util\_difference}{temp}}$ **then** ▷ Accepts candidate solution
      $current\_sol \leftarrow new\_sol$
      $current\_sol\_util \leftarrow new\_sol\_util$
      if $util\_difference \leq MIN\_IMPROVEMENT$ **then** ▷ Updates nº iterations without improvement
        $no\_update \leftarrow 0$
      else if
        then$no\_update \leftarrow no\_update + 1$
      end if
      $i \leftarrow i + 1$
      if $current\_sol\_utility > best\_sol\_utility$ **then** ▷ Updates best solution
        $best\_sol\_utility \leftarrow current\_sol\_utility$
        $best\_sol \leftarrow current\_sol$
      end if
    end if
  end while
end while

Algorithm 5 Function used to select the best neighbours from a neighbourhood generated with 6 solutions. Code available in Listing B.5

for `i in range(6)` **do**
  $new\_solution \leftarrow GenerateNewSol(current\_sol, i)$ ▷ Generates solution, different operators
  if $new\_solution == False$ **then** ▷ Failed to generate solution
    $Continue$
  end if
  $new\_sol\_util \leftarrow SolutionUtility(new\_solution)$
  if $new\_sol\_util > best\_solution\_util$ **then** ▷ Check if it is the best solution found
    $best\_solution\_util \leftarrow new\_sol\_util$
    $best\_solution \leftarrow new\_sol$
  end if
end for

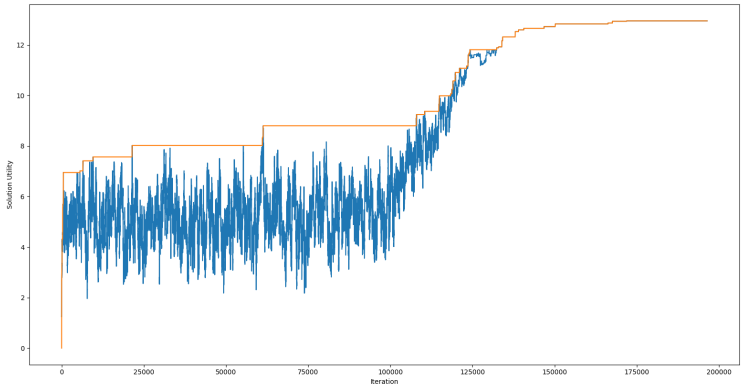


Figure 7.9: Simulated Annealing solution utility throughout the search

it is visible that any solution is accepted no matter its utility, as there are iterations of higher and lower utility, but the solution's utility average remains the same. In the central part of the search, the algorithm keeps having higher and lower utility iterations, but the overall utility average tends to increase, and the algorithm finds new best solutions more often than in the initial phase. In the third and last phase, the solution refinement phase is visible, where the algorithm accepts almost exclusively solutions of higher utility.

7.8.3 Tabu-Search Algorithm

The Tabu-search proposed in this dissertation is complex, resulting from the combination of distinct implementations and methods regarding Tabu-search analysed in the literature. The improvements done to the basic implementation of the TS algorithm were reportedly improving its search performance. This algorithm also requires a new method of generating new solutions with different operators compared to the past two methods: Hill-Climbing and Simulated Annealing. This implementation contains strategies to address the diversification, where the solutions will be generated prioritising less frequent solution elements, and intensification phases, where the search might restart from an elitist solution.

The proposed implementation comprises three different memory structures: two short-term memory structures (tabu-lists) to store the tabu operations for the next N-iterations and one long-term memory structure to store the frequency of solution elements. These structures are initialised with the algorithm and updated during the search phase. The tabu list size used was equal to half of the size of the complete list of economic operators; this represents how many iterations a certain operation will be tabu. The implementation of the two tabu lists was due to the two operation types: exchange economic operators and add/remove operations. The first list, a two-dimensional matrix, will store the tabu moves regarding exchanges and the second tabu-list, a one-dimensional list, the moves regarding add/remove of economic operators. Every time two operators are exchanged to obtain a new solution, the corresponding table entry is updated with the current iteration summed with the tabu list size. This methodology avoids the need to update the iteration counter inside

the tabu lists in every iteration; when a new move is selected, the algorithm only checks the corresponding cell in the matrix, and, if the current iteration is greater than that cells' value, that move is considered valid. If a sequence of two operators is exchanged (2*-OPT), two entries are updated, the one corresponding to the two first elements and the two second elements of both sequences. Every time a new entry is to be updated, a method sorts the two economic operator's ids, such that the same cell is always updated, in cases like A exchanging with B and B exchanging with A.
When a new operator is added or deleted from an inspection route, the entry corresponding to its ID is updated in the tabu-list with the current iteration counter plus the tabu-list size. The frequencies table is updated every time the algorithm updates the current solution, using the economic operators' ids composing the solution; the update consists in summing one unity to the current frequency of each economic operator. If two operators are removed or added (2*-OPT), the two corresponding entries are updated in the tabu list.

```
1  def buildTabuLists(indexes): #Build tabu lists
2      tabus = dict()
3      temp_list = [[0,0] for i in range(len(indexes))]
4      tabus['change'] = [temp_list.copy() for i in range(len(indexes))]
5      tabus['add'] = temp_list.copy()
6      return tabus
7
8  def buildFrequencyTable(indexes): #Build Frequency table
9      frequency_table = np.zeros((len(indexes),), dtype=int)
10     return frequency_table
11
12 def updateFrequencyTable(frequency_table, solution, indexes): #Update frequency
       table
13     for route in solution:
14         for op in route[1:-1]:
15             frequency_table[indexes[op['id']]] = frequency_table[indexes[op['id']]]
                   + 1
16     return 0
17
18 def setTableValue(id_line, id_col, indexes, tabus, value, objective_function):#
       Update tabu-list exchange
19     if id_line > id_col:  #Sort the Ids
20         temp = id_line
21         id_line = id_col
22         id_col = temp
23     line = indexes[id_line]
24     col = indexes[id_col]
25     tabus[line][col] = [value, objective_function]
26     tabus[col][line] = [value, objective_function]
27     return 0
28
29 def setTableValueAdd(id_line, indexes, tabus, value, objective_function):#Update
       tabu-list add/delete
```

```
30      line = indexes[id_line]
31      tabus[line] = [value, objective_function]
32      return 0
```

Listing 7.2: Methods to create and update the tabu-lists and frequency tables

The solution generation method had to be changed for this algorithm. This new method generates and evaluates the utility of a neighbourhood with 100 elements obtained randomly from the current solution using a set of operators. Six operators are used with an equal frequency for this purpose. The solution generator doesn't always generate 100 new solutions because some operators might fail to execute. Every successful operator generates a new solution even if it performs a move that is considered tabu. The algorithm signalises solutions generated by tabu moves, and they can only be selected as the best neighbour if they fulfil one aspiration rule. The algorithm will choose the solution with the highest utility from all the valid solutions generated; valid solutions are not originated by a tabu move or fulfil an aspiration criterion. After choosing the best solution, the tabu-lits are updated with the move performed to create that solution and the frequency table with the economic operators composing that solution. Although the operators remain the same, they had to be changed and adapted to the TS algorithm. The operators used in the HC and SA algorithms don't change the complete list of operators. For example, when the fourth operator is used on a solution, a random inspection route gets scheduled with an extra-economic operator from the complete list of economic operators. This economic operator isn't removed from the list of operators, meaning future moves on other routes can use it and, although unfeasible and with a penalty, a solution could inspect the same economic operator by the same or different inspection routes. In the TS algorithm, each economic operator only appears once in the system, either in one of the inspection routes or in the complete list of economic operators. Operations like adding an economic operator to one of the routes remove the same economic operator from the full list of economic operators; the complete list of economic operators behaves almost like an inspection route, with all the economic operators in the database that are not scheduled for inspection by one of the brigades in the current solution.

Algorithm 6 Tabu-Search solution generation from the Neighbourhood. Code available in Listing B.6

for `i in range(`*NEIGHBOURS*`)` **do**
  selected_operator ← *i%DIFF_OPERATIONS*
  new_solution ← *GenerateNewSol*(*current_sol*, *selected_operator*) ▷ Generates solution
  if *new_solution* == *False* **then** ▷ Failed to generate solution
    Continue
  end if
end for

The algorithm considers a new solution valid if it was not generated using a tabu move or fulfils one of the Aspiration criteria. In this dissertation, two Aspiration criteria were implemented and allow for tabu moves to be accepted [7]. The first Aspiration criterion accepts a tabu solution

if its utility is higher than the best solution's utility. The second Aspiration criterion verifies if the utility of a new solution generated by a tabu move is higher than the utility of the solution generated by the same move in a past iteration of the search. For this purpose, when updating the tabu-lists, one must store the utility value of the solution corresponding to that move.

Algorithm 7 Tabu move verification and use of the Aspiration criteria. Code used available in Listing B.7

if $(!was\_tabu) \vee (new\_sol\_util > best\_sol\_util) \vee (new\_sol\_util > old\_tabu\_sol\_util)$ **then** ▷ Checks if a move is accepted
  if $new\_sol\_util > best\_neighbor\_util$ **then** ▷ checks if it is the best neighbor found
    $best\_neighbor\_util \leftarrow new\_sol\_util$
    $best\_neighbor \leftarrow new\_sol$
  end if
end if

The algorithm will enter the diversification phase during the search process if it doesn't find a new best solution for a defined number of iterations. In this implementation, this number was set to 30000 iterations. When this threshold is reached, the algorithm changes the utility value of every solution generated in a neighbourhood. A discount will be subtracted to every solution's utility depending on how frequent the economic operators composing it. The frequency table is used to obtain the economic operator's frequency. The discount is calculated by dividing the sum of frequencies of one solution by the table's total frequencies [7.3].

```
1
2  def getOperatorsFrequency(frequency_table, solution, indexes):
3      total_frequency = 0
4      for route in solution:
5          for op in route[1:-1]:
6              total_frequency += frequency_table[indexes[op['id']]]
7      return total_frequency
8  ##########################################################
9  #(...)
10         #Calculate Utility According to frequencies
11         solution_combined_frequency = getOperatorsFrequency(frequency_table,
               new_solution, indexes) #Gets the frequency sum
12         solution_combined_frequency = solution_combined_frequency / sum(
               frequency_table) #Gets the frequency ratio
13         new_solution_utility = new_solution_utility - solution_combined_frequency
               #Subtracts the frequency
14         new_solution_utility = new_solution_utility - penality #apply penality when
                violating contraints
15 #(...)
```

Listing 7.3: Implementation of the Diversification phase using the frequency table

This implementation considers both feasible and unfeasible solutions. The search can sometimes get stuck in the unfeasible region, and it might be hard or time-consuming to get back to the feasible search space. This dissertation used an intensification strategy based on elitist solutions to address the previous problem [7.4]. Elitist solutions are solutions that were explored and save in long-term memory for later use. After being for a defined number of iteration in the unfeasible region, the algorithm can restart its search from one of the elitist solutions kept in memory. Elitist solutions are always feasible solutions, meaning that by restarting the search, the algorithm will be back to the feasible region. The approach developed consists of storing ten elitist solutions, and the search will restart from one if, during 20000 iterations, the search couldn't leave the unfeasible region. Every time a new best solution is found, it is added to the elitist solution list. When there are ten elitist solutions, the next ones are added, and the old ones are deleted. Elitist solutions are selected randomly but weighted according to the utilities of each elitist solution.

```
1  def addElitistSolution(elitist_solutions, solution, solution_utility):
2      if len(elitist_solutions) >= 10: #Allows 10 elitist solutions
3          elitist_solutions.pop(0)
4      elitist_solutions.append([solution, solution_utility])
5      return 0
6
7  def selectElitistSolution(elitist_solutions):
8      #elitist_solution = [solution, solution_utility]
9      total_utility = 0
10     solution_weights = []
11     for solution in elitist_solutions:
12         total_utility += solution[1] #gets the total utility
13     for solution in elitist_solutions:
14         solution_weights.append(solution[1]/total_utility)
15     solution = random.choices(elitist_solutions, weights=solution_weights,
           cum_weights=None, k=1)[0]
16     #The solution is selected randomly according to the weights
17     return solution[0]
```

Listing 7.4: Implementation of the Diversification phase using the frequency table

Figure 7.10 represents the search evolution of the Tabu-Search method. Since this method selects the best solution out of the neighbourhood of the current solution, worse solutions can be accepted. The graph shows the utility of the current solution throughout the search, and it is possible to see negative utility variations. The total best solution is always kept in memory and is updated with the current solution if a new best solution has been found.

7.8.4 Large Neighborhood Search

The Large Neighborhood Search (LNS) was implemented jointly with a Tabu-search implementation. This algorithm minimises a large neighbourhood of solutions into a small neighbourhood

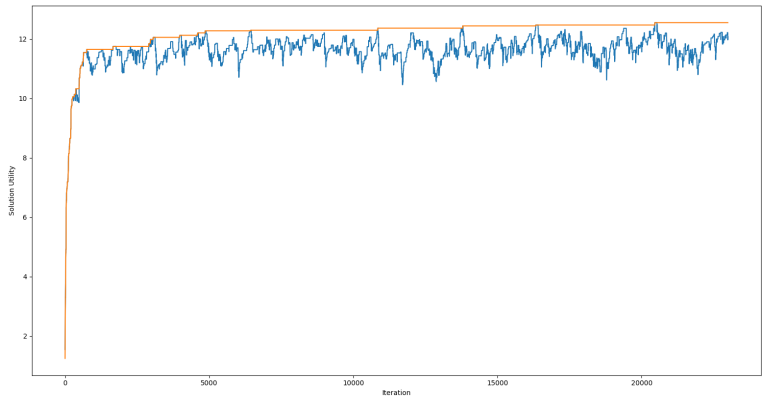


Figure 7.10: Tabu-Search solution utility throughout the search

by fixing specific parts of the solution. Search methods tend to be less efficient and precise over large neighbourhoods as the search space is more extensive, increasing the possibility of getting stuck into a local maximum. In this problem's context, the building blocks of a solution are the inspection routes since they are independent. A solution can be easily divided into its composing inspection routes.

In each iteration, the algorithm fixes a number of inspection routes, meaning they will not take part in the optimisation process in that iteration. Only two inspection routes will be optimised in each algorithm's iteration, significantly reducing the neighbourhood's size and facilitating the search for problems with many brigades [7.5]. For this purpose, the individual utility of all the routes composing one solution is calculated using the utility function described in section 7.7. These inspection routes are then sorted based on their utility, and the ones with the two lower utilities are selected to be optimised. The optimisation process uses the implementation of the tabu search described in section 7.8.3. The Tabu-search algorithm optimises and solves an artificial problem instance with only two inspection routes and returns the best solution found during the search. Ten thousand iterations are used as the stopping criterion for the Tabu-search execution. The solution returned by the TS (two inspection routes) is then appended with the remaining routes that were fixed for that LNS iteration rebuilding the original solution. The new solution's utility doesn't need to be evaluated as it is guaranteed to be at least as good as the solution before optimisation; the tabu-search never returns a worse solution compared to the initial solution.

This LNS implementation always starts with a feasible solution. The initial solution is tested on its feasibility and, if it fails, a feasible solution needs to be found before the LNS can start. If the initial solution is feasible, the algorithm will start its execution. Otherwise, a tabu search algorithm will optimise the initial solution for a defined number of iterations hoping to find a feasible solution. If no feasible solution is found in N iterations, the search terminates.

```
1 def optimiseNeighborhood(...):
2     min_objective_value_index = dict()
```

```
3     min_objective_values = []
4
5     for i, route in enumerate(current_solution):
6         current_solution_utility, last_time, penality = algorithms_functions.
              solution_utility(...) #calculates the inspection routes utility
7         min_objective_values.append(current_solution_utility)
8         min_objective_value_index[current_solution_utility] = i
9
10    min_objective_values.sort() #sorts the inspection routes according to their
          utility
11    index1 = min_objective_value_index[min_objective_values[0]]  #route 1 to
          improve
12    index2 = min_objective_value_index[min_objective_values[1]]  #route 2 to
          improve
13
14    neighborhood = [current_solution[index1], current_solution[index2]] #
          neighborhood to optimise
15    neighborhood_start_time = [starting_time[index1], starting_time[index2]] #
          neighborhood start times
16
17    best_utility, new_neighborhood_solution, best_penality, new_operators =
          tabu_search.algorithm(...) #tabu search algorithm
18
19    current_solution[index1] = new_neighborhood_solution[0] #builds the new
          solution
20    current_solution[index2] = new_neighborhood_solution[1] #builds the new
          solution
21
22    current_solution_utility, last_time, penality = algorithms_functions.
          solution_utility(...) #calculates new solution's utility
23    return current_solution, current_solution_utility, new_operators
```

Listing 7.5: LNS shrinks one solution's neighbourhood and solves a minor optimisation problem.

7.8.5 Summary

All the implementation steps described in this chapter originated a system capable of generating disruptions to a set of inspection routes, composing an operational plan for one workday. The operational plan is optimised using four optimisation algorithms, subject to a utility function adjustable by a business expert. All the disruption prototypes considered in this work represented the real-world scenario of ASAE's inspection operations and were proposed and validated by business experts. The following chapter contains several experiments to test and compare the system's capabilities in solving the disruptions, with results and conclusions.

Chapter 8

Results and Analysis

8.1 Introduction

To obtain results, conclusions and other relevant pieces of information about the system developed in this dissertation's scope, it was tested in a series of controlled scenarios. The main objective of this approach is to analyse how the different algorithms react to the different disruption types and smaller or larger disruptions. It is also an objective to obtain conclusions about each method performance and scalability with an increasing number of inspection routes. The execution time of each method will also be analysed, searching which one delivers the best solution in less time interval, making it more suitable to use in the real-life scenario. The utility function will also be tested with different weights between its three components and how each algorithm reacts to it.

This dissertation will perform all the executions in the same problem instance, explained next and adapted from a real-life scenario. The problem instance analysed can be considered a sub-problem of the real-life environment with fewer inspection routes and economic operators. The initial depot used for all the runs will be the same operational unit, and all the economic operators considered for the executions belong to its area of operation (this operational unit is the one responsible for their inspections in the real-life scenario) and encapsulated in a smaller area of the Portuguese territory. The previous will indirectly encapsulate the problem in a smaller Portuguese region, where the economic operators are located. The fixed problem instance used for all the system runs can be defined as follows:

- The operational unit considered is the Unidade Operacional III - Mirandela, belonging to the district of Bragança, county of Mirandela, parish of Carvalhais. Its located in Quinta do Valongo, Vila Nova das Patas with the postal code 5370-087. It is georeferenced in ASAE's databases with the coordinates (41.5140871, -7.1835093), latitude and longitude, respectively.

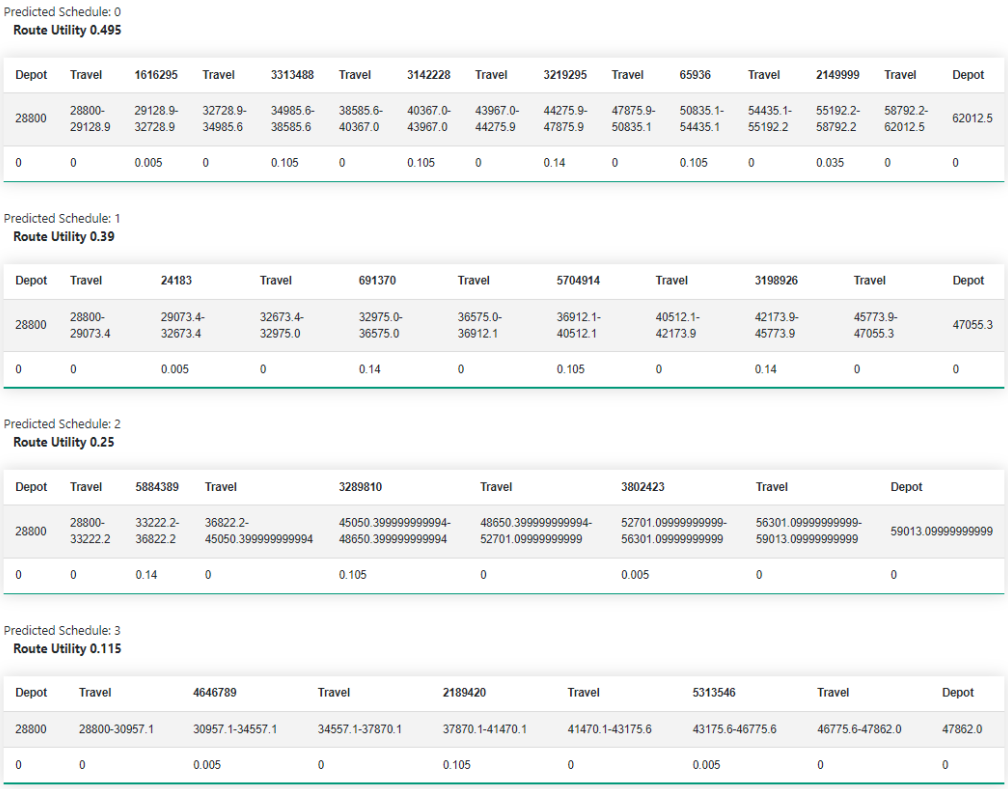
- All the economic operators used in this system belong to the previous operational unit and correspond to the complete list of economic operators stored in the database with the needed

parameters filled with valid information. Removing the economic operators that don't satisfy the requirements, this problem instance considered 532 different economic operators with distinct pairs of coordinates and corresponding utilities. Table A.1 shows the distribution of economic operators among the distinct locations.

- The brigades start working at 8:00 in the morning (28800 seconds) and finish at 18:00 (64800 seconds). The brigades work for 10 hours in a row, starting and arriving at the depot (operational unit) at the end of the workday. The work is continuous, meaning this simulation doesn't allow any breaks, for example, lunch breaks. The simulation takes time on a Wednesday, 8 September 2021.
- In some experiments, the economic schedules are randomly generated, as explained in section 7.6. Other executions use an artificially generated schedule for all the system's economic operators. This schedule represents an economic operator that is opened 24/7.
- As the initial solution, four inspection routes will be considered for all the test cases. The initial inspection routes were generated by Barros [27] approach using several metaheuristic based methods. These inspection routes compose an operational plan, and before disruptions are generated, this plan is always feasible. Information about the inspection routes can be found in Table 8.1. The initial schedule forecasted for this solution can be found in Image 8.1, where each table indicates the predicted steps for each inspection route with the respective timestamps.
- All tests used an inspection time of 1 hour (3600 seconds) for all the economic operators in the system. The disruption Generator can generate disruptions that will affect these inspection times.

Since the algorithm implementations generate different neighbourhood sizes in each iteration, using the iteration counter as the stopping criteria for the algorithms won't be fair. One iteration of the hill-climbing algorithm only generates one neighbour, while an iteration from tabu-search can generate at most one hundred neighbours. The approach used to perform a fair comparison among the algorithms set the stopping criteria as a defined target execution time. Contrarily to the other algorithms, the simulated annealing doesn't accept the execution time as the stopping condition because its execution relies on the cooling schedule. The temperature decrement rule used in this dissertation involves the number of Markov chains, which is a stochastic variable and can change depending on each execution. Instead of using the execution time as the stopping condition, the iteration counter was used. A value for the target iterations was manually selected to keep the execution time close to the target of the other algorithms. The Large neighbourhood search can have execution times larger than the target for the test since the current iteration cant be stopped in the middle. Therefore the algorithm will finish the current iteration and return the best solution after.

After the execution of every test scenario, a solution is produced. Several metrics are measured and registered: the average utility of economic operators inspected by that solution (*avg_UA*),



Predicted Schedule: 0
Route Utility 0.495

Depot	Travel	1616295	Travel	3313488	Travel	3142228	Travel	3219295	Travel	65936	Travel	2149999	Travel	Depot
28800	28800-29128.9	29128.9-32728.9	32728.9-34985.6	34985.6-38585.6	38585.6-40367.0	40367.0-43967.0	43967.0-44275.9	44275.9-47875.9	47875.9-50835.1	50835.1-54435.1	54435.1-55192.2	55192.2-58792.2	58792.2-62012.5	62012.5
0	0	0.005	0	0.105	0	0.105	0	0.14	0	0.105	0	0.035	0	0

Predicted Schedule: 1
Route Utility 0.39

Depot	Travel	24183	Travel	691370	Travel	5704914	Travel	3198926	Travel	Depot
28800	28800-29073.4	29073.4-32673.4	32673.4-32975.0	32975.0-36575.0	36575.0-36912.1	36912.1-40512.1	40512.1-42173.9	42173.9-45773.9	45773.9-47055.3	47055.3
0	0	0.005	0	0.14	0	0.105	0	0.14	0	0

Predicted Schedule: 2
Route Utility 0.25

Depot	Travel	5884389	Travel	3289810	Travel	3802423	Travel	Depot
28800	28800-33222.2	33222.2-36822.2	36822.2-45050.399999999994	45050.399999999994-48650.399999999994	48650.399999999994-52701.09999999999	52701.09999999999-56301.09999999999	56301.09999999999-59013.09999999999	59013.09999999999
0	0	0.14	0	0.105	0	0.005	0	0

Predicted Schedule: 3
Route Utility 0.115

Depot	Travel	4646789	Travel	2189420	Travel	5313546	Travel	Depot
28800	28800-30957.1	30957.1-34557.1	34557.1-37870.1	37870.1-41470.1	41470.1-43175.6	43175.6-46775.6	46775.6-47862.0	47862.0
0	0	0.005	0	0.105	0	0.005	0	0

Figure 8.1: Schedule calculated for the initial solution with 4 inspection routes

the utility of economic operators in the best solution found (*max_UA*), the average value of the utility function (*avg_UF*), the utility function value of the best solution found (*max_UF*), the average similarity ratio between the initial and final solutions (avg_Sim), the maximum similarity ratio (max_Sim), the average time to get the best solution (*avg_TS*), the minimum time to get the best solution(*min_TS*) the average iteration counter to get the best solution (*avg_ite*), the average number of economic operators in the complete solution (*avg_OP*).

The tests were executed using an Intel i7 8700, with 16 GB RAM, running the latest stable version of Windows 10.

8.2 Algorithm Comparison

This first test case has the main objective to make a raw comparison between the four algorithms. For these purposes, the schedule used will be an artificially generated schedule corresponding to an operator that is always open. The utility function used in this test will be merely the economic operator's utility [Section 7.7.1], and the other utility function components will be given a weight of 0%. The simulation time used was the same as the inspection routes start time, 28800 seconds. The only disruptions generated for these tests were travelled time and inspection time disruptions, always using a Gaussian distribution with the same standard deviation (σ = current travel time

Table 8.1: 4 Inspection Routes used for the testing

Inspection Route ID	Start Time	Total Utility	Nº Econ. Operators	Last Time
7	28800	0.495	6	62012.5
8	28800	0.39	4	47055.3
9	28800	0.25	3	59013.09
10	28800	0.115	3	47862.0

Table 8.2: Tests identification and parameters (Experiment 1)

ID			Algorithm	Stop Condition	Stop Value
1-HC-30	2-HC-30	3-HC-30	Hill-Climb	Execution Time	0.5
1-SA-30	2-SA-30	3-SA-30	Simulated Annealing	Markov Chains	variable
1-TS-30	2-TS-30	3-TS-30	Tabu-Search	Execution Time	0.5
1-LNS-30	2-LNS-30	3-LNS-30	Large Neighborhood Search	Execution Time	0.5
1-HC-2	2-HC-2	3-HC-2	Hill-Climb	Execution Time	2
1-SA-2	2-SA-2	3-SA-2	Simulated Annealing	Markov Chains	variable
1-TS-2	2-TS-2	3-TS-2	Tabu-Search	Execution Time	2
1-LNS-2	2-LNS-2	3-LNS-2	Large Neighborhood Search	Execution Time	2
1-HC-4	2-HC-4	3-HC-4	Hill-Climb	Execution Time	4
1-SA-4	2-SA-4	3-SA-4	Simulated Annealing	Markov Chains	variable
1-TS-4	2-TS-4	3-TS-4	Tabu-Search	Execution Time	4
1-LNS-4	2-LNS-4	3-LNS-4	Large Neighborhood Search	Execution Time	4

/ 9). In this experiment, the independent variables are the execution time and the number of inspection routes. The execution time used was thirty seconds, two and five minutes, while the number of brigades was two, three and four. The execution with two brigades used inspection routes 7 and 8, the execution with three brigades used inspection routes 7,8,9, and the execution with four brigades used inspection routes 7,8,9,10. All the tests and corresponding parameters are shown in Table 8.2. A string identifies each test. The first sub-string identifies the problem instance, the second represents the algorithm used, and the last represents the execution time. The problem instance 1,2, and 3 illustrates a setup with 2, 3, and 4 inspection routes, respectively. The experiment using the Large Neighborhood Search with two inspection routes was discarded since the results were expected to be the same as the Tabu-search; the destroy method wouldn't select a smaller portion of the neighbourhood but the same two inspection routes.

This first experiment consisted of 36 different test instances with three executions for each test instance. The system implemented in this dissertation has been executed a total of 108 times, and the results can be found in Table 8.3. The SA was tested with 12 neighbours and TS with 100 in all test instances. LNS used 50 neighbours in the 30 seconds and 2 minutes executions, while 100 in the four minutes run. Regarding the SA, the stopping criteria used was the number of Markov chains. These number was adjusted to 200000, 400000, and 900000 in the problem instances scheduled for an execution time of 30 seconds, 2 minutes and 4 minutes, respectively.

Table 8.3: Test Results (Experiment 1). UA - sum of economic operators utilities; Sim - Similarity ratio; TS - search execution time; OP - number of economic operators; ite - number of iterations

ID	avg_UA	max_UA	avg_TS	min_TS	avg_OP	avg_ite
1-HC-30	7.87	8.28	11.45	5.92	17.67	669271.62
1-SA-30	8.25	8.30	21.83	17.16	17.33	162320.48
1-TS-30	8.24	8.30	13.33	4.95	17.00	8575.27
1-LNS-30	-	-	-	-	-	-
1-HC-2	7.28	7.78	17.16	9.37	18.00	2309964.81
1-SA-2	8.22	8.29	66.87	65.35	16.33	775867.34
1-TS-2	8.30	8.30	10.50	4.17	18.00	34136.96
1-LNS-2	-	-	-	-	-	-
1-HC-4	7.26	7.58	37.49	3.95	18.33	5533169.67
1-SA-4	8.29	8.29	194.38	117.59	17.00	1479542.20
1-TS-4	8.30	8.30	25.60	2.13	17.67	52362.12
1-LNS-4	-	-	-	-	-	-
2-HC-30	9.90	10.66	22.74	22.06	27.67	501589.01
2-SA-30	10.69	10.72	29.15	22.06	25.00	155668.92
2-TS-30	10.53	10.60	19.23	9.87	27.00	6410.68
2-LNS-30	10.53	10.61	32.84	31.33	26.00	2.34
2-HC-2	10.52	10.71	34.78	11.61	27.00	1504300.34
2-SA-2	10.75	10.76	100.43	81.63	25.00	669831.17
2-TS-2	10.79	10.83	35.49	20.09	26.33	27399.83
2-LNS-2	10.72	10.87	104.57	77.31	26.00	8.76
2-HC-4	10.19	10.30	31.39	13.96	28.00	4108895.56
2-SA-4	10.74	10.76	147.56	133.57	25.00	1062071.23
2-TS-4	10.77	10.83	141.06	102.23	25.67	54383.74
2-LNS-4	11.50	12.77	153.64	97.35	28.33	8.72
3-HC-30	12.46	12.58	19.85	15.43	37.33	405880.32
3-SA-30	12.91	13.03	33.95	30.02	35	143106.28
3-TS-30	12.71	12.86	20.58	18.79	35.33	5336.56
3-LNS-30	12.80	12.91	32.44	31.74	35	2.00
3-HC-2	12.83	13.08	78.32	50.42	36.33	1699673.92
3-SA-2	12.93	12.99	109.19	101.01	33.66	520236.81
3-TS-2	12.81	12.87	33.50	23.76	36.00	19717.53
3-LNS-2	12.95	12.99	95.07	77.82	34.33	8.67
3-HC-4	12.53	12.58	78.81	13.39	36.00	3456250.65
3-SA-4	13.17	13.28	148.90	146.28	35.33	805707.45
3-TS-4	12.97	13.20	202.92	163.66	35.33	42549.12
3-LNS-4	13.92	15.66	116.60	31.70	34.33	7.33

8.2.1 Result analysis

All the algorithms performed relatively well in all the problem instances using all the running times defined. The solutions obtained by each algorithm were close in their utility, without any

algorithm performing significantly worse than the others.

Regarding the run times, the simulated annealing algorithm performs the worse since it spends a significant time in the exploratory phase, where the algorithm jumps in the search space without investing its time finding a local optimum. The algorithm has low-temperature values in the last iterations, and only then it improves the current solution, finding the nearby local maximum. Therefore the time taken to find the best solution is usually the same as the execution time. The Hill-Climbing algorithm finds the best solution in the shortest execution time, scoring the best execution time in five out of nine tests. This algorithm is the one that converges to a solution faster. The shorter average time to find a solution is explained because the algorithm gets trapped in a local maximum and cant improve the solution further. The Tabu-Search comes next, scoring the best execution time in four out of nine tests.

Since during this experiment, the utility function was only dependent on the utility values, this is the primary metric that will be taken into account for the quality of a solution. Since this approach targets consistency in finding the best solution, the metric average solution utility is more important than finding the best solution during the three runs. The algorithm that performs the worse in terms of solution quality is the Hill-Climbing having the lowest average solution utility in eight out of the nine tests. Besides that, Hill-Climbing can still find reasonable solutions since its execution depends on stochasticity, and it can sometimes get trapped in a good local maximum. For problem instances with two inspection routes, the TS algorithm best finds the highest utility solutions and the highest average solution utility. For larger problem instances with more inspection routes, the LNS revealed to be the best algorithm to find the best solutions consistently.

The number of operators composing a solution was also analysed for each algorithm, given that it contributes to the quality of one solution; an operational plan with fewer inspections is expected to be less vulnerable to disruptions. The simulated annealing is the algorithm that can output a good solution utility wise compared to the others but using fewer inspections. SA finds the solutions with fewer inspections in eight out of the nine tests instances. The Hill-Climbing is the algorithm that needs more economic operators, performing the worse in all the tests, even delivering solutions with less utility when compared to the others.

By analysing the results, it can be concluded that longer run times produce solutions of higher utility. The only exception is the Hill-Climbing algorithm that appears to converge in a shorter time period, which means that more iterations won't affect the outcome. The LNS is the algorithm that takes the most advantage of additional computational time. The difference between the solution found by two and four minutes of execution is close to one unit of utility value.

8.3 Disruption Types Comparison

This section aims to test how the different algorithms react to the various disruptions considered in this work. The schedule used for the tests was artificially generated and is equivalent to an economic operator that is always opened. The utility function was composed of two components:

a weight of 0.66 for the utility sum of all the inspected operators (Section 7.7.1) and 0.33 for the solution's similarity (Section 7.7.2). The simulation time used was the same as the inspection routes start time, 8:00 AM (28800 seconds). For the tests executed in this section, the algorithms execution time was topped at four minutes, and the simulated annealing was manually adjusted to run in this same time interval. The operational plan was equal for all the tests and composed of four inspection routes, with Ids 7, 8, 9,10. All the disruption types were tested in this section: the inspection and travel times disruptions were generated using a $\delta = Current\_Travel\_Time/5$, the vehicle and inspection breakdowns were set to one occurrence in the whole operational plan, the utility changes took action over the economic operators belonging to classes 'III', 'V', 'VI', and the emergency inspections were set to two occurrences in the whole operational plan. The new economic operators that will be part of the emergency inspection are selected randomly in all the tests. Since they belong to the same geographic area, it is expected for the algorithms to include them in the routes with similar effectiveness. All the tests and corresponding parameters are shown in Table 8.4. The first digit identifies the experiment number, the second digit identifies the algorithm, and the third represents the test number.

This section consisted of 24 different test instances with the average values gathered from three executions for each test instance. The metrics analysed in this section are: the average utility function values, the max utility function value, the average utility sum of all the economic operators, the max economic operator's utility sum, the average similarity ratio, the best similarity ratio, the average time to find the best solution and the average economic operator's quantity in the complete solution. The similarity ratio (Section 7.7.2) indicates how similar the initial operational plan is to the new one after the re-optimisation. This ratio represents the average similarity between each route in the final solution and its corresponding route on the initial solution.

The approach proposed was executed 72 times, and the results are shown in Table 8.5. The SA was tested with 12 neighbours, and the TS and LNS algorithms with 100 neighbours in all test instances.

8.3.1 Result analysis

The generality of the algorithms solved all the problem instances with reasonably similar solutions utility wise, indicating that good quality solutions are being found. The only algorithm that failed to deliver a solution was the hill climb in the emergency inspection disruption type.

All the methods besides the SA reacted well with the utility function that also depends on the similarity ratio. They provide solutions with similar utility function values, which come from the combination of the similarity ratio and the utility gathered by inspecting all the economic operators composing the operational plan. Contrarily, the SA doesn't provide solutions with a similarity ratio. The only positive similarity ratio was on the emergency inspection test, but this is due to the emergency inspections themselves since they have to be accomplished for the solution to be feasible. It is therefore expected for these economic operators to be both in the initial and the reoptimized solution. This behaviour has to do with the first two phases of SA search. Since the algorithm can accept worse solutions, the economic operators composing the initial solution are

Table 8.4: Tests identification and parameters (Experiment 2)

ID	Algorithm	Disruption Type	Dis Param	Value
HC-IT	HC	Inspection Time	Disruption Strength	5
SA-IT	SA	Inspection Time	Disruption Strength	5
TS-IT	TS	Inspection Time	Disruption Strength	5
LNS-IT	LNS	Inspection Time	Disruption Strength	5
HC-TT	HC	Travel Time	Disruption Strength	5
SA-TT	SA	Travel Time	Disruption Strength	5
TS-IT	TS	Travel Time	Disruption Strength	5
LNS-IT	LNS	Travel Time	Disruption Strength	5
HC-VB	HC	Vehicle Breakdown	Nº Vehicles	1
SA-VB	SA	Vehicle Breakdown	Nº Vehicles	1
TS-VB	TS	Vehicle Breakdown	Nº Vehicles	1
LNS-VB	LNS	Vehicle Breakdown	Nº Vehicles	1
HC-UC	HC	Utility Changes	Econ. Operator class	III,V,VI
SA-UC	SA	Utility Changes	Econ. Operator class	III,V,VI
TS-UC	TS	Utility Changes	Econ. Operator class	III,V,VI
LNS-UC	LNS	Utility Changes	Econ. Operator class	III,V,VI
HC-IB	HC	Inspection Breakdown	Nº Inspections	1
SA-IB	SA	Inspection Breakdown	Nº Inspections	1
TS-IB	TS	Inspection Breakdown	Nº Inspections	1
LNS-IB	LNS	Inspection Breakdown	Nº Inspections	1
HC-EI	HC	Emergency Inspection	Nº Inspections	2
SA-EI	SA	Emergency Inspection	Nº Inspections	2
TS-EI	TS	Emergency Inspection	Nº Inspections	2
LNS-EI	LNS	Emergency Inspection	Nº Inspections	2

removed. The 0.33 weight given to the similarity ratio in the utility function is not enough for this algorithm to find them back instead of adding others with higher utility values. As expected, this algorithm has the higher values for the economic operators' utility component of the utility function in all the tests since having no similarity ratio creates a search bias to gather the most utility possible. Other algorithms balance the utility gathering with keeping the solution relatively similar to the initial operational plan. The similarity ratio of the other algorithms, rounding the 30% similarity, could also be higher in a real scenario. The routes used as the initial solution were generated and stored in the database using a different approach with different algorithms and economic operators. The system used in this dissertation uses a higher quantity of economic operators, meaning the routes from the initial solution are sub-optimised. Therefore, this implementation will further optimize the inspection routes at the cost of losing some similarities. The similarity weight could also be adjusted, giving more utility to solutions closer to the original operational plan. Regarding the average time to get the solution, the Hill-Climbing is the algorithm that quickly finds the best solution, while the simulated annealing is the one that performs worse on this metric. The Tabu-search and the Large neighbourhood search have similar average time intervals to find the best solution.

The tests that have the most relevant results are the ones involving emergency inspections. The Hill-Climbing failed to solve two out of the three problem instances, outputting unfeasible solutions with associated penalties. The algorithm seems to get stuck in the initial solution, having difficulty progressing in the search. The similarity values support this conclusion since an average of 0.69 indicates that the solutions didn't advance much from the initial solution (test HC-EI). The emergency inspections can sometimes be far from the initial route course and, therefore, difficult to be added to an inspection route while maintaining the feasibility of the operational plan. An algorithm like Hill-Climbing is not suitable to solve such disruption since it doesn't allow worse solutions, not allowing the emergency inspections to be readjusted freely in the plan. An unfeasible solution might be needed so that the algorithm can reach the feasible solution space. All the tests with emergency inspections have a substantially higher utility since the two emergency inspections alone represent 200 of utility value. The solutions obtained in these tests also have a higher similarity ratio, as the two emergency operators have to be kept in the solution.

Inspections and vehicle breakdowns will condition the inspection routes, and no further operator can be inspected by that route, implying that no extra utility value can be gathered. The tests containing these types of disruptions have a lower utility since only three brigades are working, not the usual four from the other tests. The results show that tests ran with the influence of utility changes disruption have a higher utility average when compared to the other ones, similarly for all the four algorithms. Since this disruption type consists of increasing the utility of certain economic operators which belong to a predefined class, it is expected for the algorithms to find a solution with higher utility. The solutions obtained by solving a problem instance with utility changes usually contain more economic operators belonging to the affected economic operators' classes.

The current implementation of the LNS can influence the algorithm's performance to solve emergency inspection disruptions. LNS reduces the solution's neighbourhood to only two inspection routes, optimizing them using a TS algorithm. The two inspection routes with less utility value are selected from the operational plan, meaning only two routes can be optimized concurrently. The economic operators corresponding to the emergency inspections cannot be exchanged between routes as easily compared to other implementations. Depending on the new inspection location, it might be better if this inspection is moved from one route to another, for instance, if it is close to the path of an existing inspection route. As expected, the results from the LNS over an emergency inspection disruption are sightly worse when compared to the TS and HC approaches. These results are expected to be worse when using more inspection routes and emergency inspections.

8.4 Full Conditions Comparison

This third section has the main objective of testing the complete approach developed in this dissertation by running the system in several tests. This section uses the schedule generation method implemented, randomly generating the schedule for each economic operator, considering its type

Table 8.5: Test Results (Experiment 2). UF - utility function; UA - sum of economic operators utilities; Sim - Similarity ratio; TS - search execution time; OP - number of economic operators

ID	avg_UF	max_UF	avg_UA	max_UA	avg_Sim	max_Sim	avg_TS	avg_OP
HC-IT	13.09	13.24	11.55	11.77	0.22	0.29	51.96	34.33
SA-IT	12.88	12.97	12.88	12.97	0.00	0.00	160.38	34.33
TS-IT	13.51	13.67	11.15	11.26	0.34	0.35	111.09	34.00
LNS-IT	14.27	15.90	12.62	15.06	0.24	0.35	91.32	33.33
HC-TT	12.93	13.08	11.48	11.69	0.21	0.24	83.53	33.00
SA-TT	12.98	13.07	12.98	13.07	0.00	0.00	150.86	33.67
TS-TT	12.99	13.05	10.66	10.80	0.33	0.36	121.86	31.67
LNS-TT	13.04	13.31	10.80	11.03	0.31	0.33	75.69	32.00
HC-VB	10.74	10.89	9.67	10.30	0.20	0.39	18.96	25.00
SA-VB	10.76	10.87	10.76	10.87	0.00	0.00	133.48	25.33
TS-VB	10.72	10.84	9.30	9.51	0.270	0.32	35.07	24.33
LNS-VB	10.64	10.83	9.96	10.27	0.21	0.37	92.73	24.00
HC-UC	10.47	10.77	9.88	10.22	0.11	0.13	34.54	24.33
SA-UC	10.90	10.94	10.90	10.94	0.00	0.00	125.70	26.00
TS-UC	10.80	10.90	9.69	10.04	0.21	0.26	103.47	25.00
LNS-UC	10.70	10.79	9.47	9.780	0.23	0.28	77.47	24.33
HC-IB	13.57	13.95	11.75	12.51	0.26	0.291	55.49	32.67
SA-IB	12.89	12.98	12.89	12.98	0.00	0.00	184.32	34.00
TS-IB	13.77	13.94	11.58	11.62	0.31	0.33	110.72	33.33
LNS-IB	13.56	13.72	11.10	11.84	0.35	0.41	87.42	31.33
HC-EI	-44.00	212.71	204.67	211.51	0.69	0.96	6.57	28.33
SA-EI	212.94	213.29	212.43	212.77	0.07	0.07	205.08	35.00
TS-EI	212.98	213.13	210.60	210.88	0.34	0.42	94.58	32.33
LNS-EI	211.99	212.44	209.37	210.42	0.37	0.54	41.81	30.00

from the ten cluster list. The utility function specified for the tests was the same used for the previous tests in Section 8.3 – a weight of 0.66 for the utility sum of all the inspected operators and 0.33 for the solution's similarity. All the routes considered in this test start at 8:00 AM (28800 seconds), and the simulation time used was 40000 seconds, meaning the optimization will take action in the fraction of the total solution following this timestamp. In all the tests, the algorithm's execution time threshold was kept at a maximum of two minutes. The same operational plan was used for the complete test set, composed of four different inspection routes with Ids 7, 8, 9, and 10. Each algorithm was tested with the four different disruption types: vehicle breakdown, inspection breakdown, utility changes, and emergency inspection.

Along with these disruptions, and in every test, travel time and inspection time disruptions were generated with a $\delta = Current\_Travel\_Time/5$. The parameters used for each test are described in Table 8.6. The operators used for the emergency inspection were randomly selected from the nearby area. Since the simulation time was set to a higher value than the routes' start times, the solutions were partial executed, and for these tests, only the utility gathered after the

simulation time start will be considered. Therefore, it is expected that these solutions are of lower quality when compared to the experiment in Section 8.3. The emergency inspections can be generated anywhere in the routes, and sometimes, this can happen before the simulation time. If the previous happens in these tests, the system already inspected the emergency economic operator, and its utility won't be captured in the results.

This section comprised 16 tests, with results gathered from 48 executions since each test results from an average of three runs. This section uses the following metrics to compare the results: the average utility function values, the max utility function value, the average utility sum of all the economic operators, the max economic operator's utility sum, the average similarity ratio, the best similarity ratio, the average time to find the best solution and the average economic operator's quantity in the complete solution.

Table 8.6: Tests identification and parameters (Experiment 3)

ID	Algorithm	Disruption Type	Dis Param	Value
HC-VB	HC	Vehicle Breakdown	Nº Vehicles	1
SA-VB	SA	Vehicle Breakdown	Nº Vehicles	1
TS-VB	TS	Vehicle Breakdown	Nº Vehicles	1
LNS-VB	LNS	Vehicle Breakdown	Nº Vehicles	1
HC-UC	HC	Utility Changes	Econ. Operator class	III,V,VI
SA-UC	SA	Utility Changes	Econ. Operator class	III,V,VI
TS-UC	TS	Utility Changes	Econ. Operator class	III,V,VI
LNS-UC	LNS	Utility Changes	Econ. Operator class	III,V,VI
HC-IB	HC	Inspection Breakdown	Nº Inspections	1
SA-IB	SA	Inspection Breakdown	Nº Inspections	1
TS-IB	TS	Inspection Breakdown	Nº Inspections	1
LNS-IB	LNS	Inspection Breakdown	Nº Inspections	1
HC-EI	HC	Emergency Inspection	Nº Inspections	3
SA-EI	SA	Emergency Inspection	Nº Inspections	3
TS-EI	TS	Emergency Inspection	Nº Inspections	3
LNS-EI	LNS	Emergency Inspection	Nº Inspections	3

8.4.1 Result analysis

The results of each test execution are shown in Table 8.7

The analysis of this experiment's results makes clear that every algorithm is effective in solving this DRPV. The conclusions of these tests are similar to the previous experiments regarding both the algorithms and disruptions types considered. The overall utility of the solutions obtained is lower because of the simulation time used, as explained before. The exception is the emergency inspection disruption, which is now considering three emergency inspections instead of two.

Since the simulation time was selected in the middle of the route, the optimizable part of the inspection routes is now more minor, meaning that the solution similarity becomes an index with greater importance. It is easier to maintain similarity as fewer operators are to be exchanged. As

explained in Section 8.3.1, the Simulated Annealing doesn't perform well in keeping the solution similarity. Therefore, and in this experiment, this algorithm performed the worse for all the test cases. The best performing algorithm was the LNS, consistently delivering the best solutions among the implemented algorithms in all the test conditions.

Similarly to the previous sections, the Hill-Climbing was the algorithm that converges faster to the best solution, although it often gets stuck in local maximums. This algorithm wasn't able to solve the emergency inspection disruptions, finding only unfeasible solutions.

The average solution similarity is more significant when compared to the previous experiments since the optimizable part of the solution is lesser, meaning it's easier for the new solution to be similar to the initial one. The average number of economic operators composing each solution was lower since only the ones used in the optimization procedure were considered. The ones already inspected before the simulation time aren't considered in the results.

Table 8.7: Test Results (Experiment 3). UF - utility function; UA - sum of economic operators utilities; Sim - Similarity ratio; TS - search execution time; OP - number of economic operators

ID	avg_UF	max_UF	avg_UA	max_UA	avg_Sim	max_Sim	avg_TS	avg_OP
HC-VB	8.18	8.72	6.43	6.60	0.33	0.42	13.68	17.67
SA-VB	7.61	8.01	7.61	8.01	0.00	0.00	59.95	17.00
TS-VB	8.62	8.97	6.58	7,24	0.39	0.5	67.24	17.33
LNS-VB	8.75	8.95	7.68	8.51	0.20	0.41	14.91	18.33
HC-UC	8.42	8.56	7.24	7.77	0.22	0.27	10.55	18.33
SA-UC	7.69	8.08	7.69	8.08	0.00	0.00	71.59	17.33
TS-UC	8.55	8.71	6.13	6.71	0.46	0.55	31.47	16.67
LNS-UC	8.85	9.09	7.34	8.50	0.28	0.42	31.23	18.33
HC-IB	10.76	11.54	8.22	9.49	0.36	0.44	6.37	23.00
SA-IB	9.67	10.19	9.67	10.19	0.00	0.00	105.69	24.33
TS-IB	10.96	11.44	7.42	8.10	0.50	0.56	45.42	22.00
LNS-IB	12.74	13.53	11.79	12.69	0.14	0.18	14.82	23.67
HC-EI	-201.71	-123.34	201.01	201.15	0.89	0.94	0.00	21.00
SA-EI	309.71	309.82	308.39	308.46	0.19	0.21	106.50	23.67
TS-EI	275.28	309.83	271.14	306.20	0.59	0.80	22.07	21.00
LNS-EI	310.15	310.27	308.16	308.49	0.28	0.31	21.95	22.33

8.5 Summary

By analysing the results globally, almost every algorithm performed as initially expected and delivering optimised solutions after addressing the disruptions generated by the disruption generator (except the HC algorithm). The different test scenarios allowed the analysis of particular aspects of the implementation and the respective conclusions regarding each algorithm.

The Hill-Climbing algorithm is the one that converges faster, delivering a reasonable solution to the problem instance relatively quickly when compared to the other algorithms. However, the

solutions found by HC are sub-optimal when compared to the remaining algorithms, as the search usually gets stuck in local maximums. This algorithm couldn't solve the emergency inspection disruption, only finding a solution that was unfeasible.

Due to its implementation and temperature decrement rules, the Simulated Annealing is the algorithm that takes more time to return the best solution. This algorithm usually finds its best solution in the last iterations. Comparatively to the HC algorithm, the quality of the solutions found by this method is higher. Nevertheless, this algorithm has trouble keeping the solution obtained after addressing the disruptions close to the initial solution. The solutions obtained by this method have none or lower similarities compared to the initial solution.

The tabu search algorithm has the best performance balancing the most utility in the final solution with the time to find it. The solutions obtained by this algorithm to every problem instance are the best or close to the best discovered by the best performing algorithm. This algorithm also delivers solutions that are similar to the solutions before the disruption was generated.

The Large Neighborhood Search is the method with the best performance in most test instances, almost always delivering the best solution utility compared to the other methods. This algorithm provides solutions with higher solution similarity. Nevertheless, it uses more computation time on average to find the best solution when compared to the TS algorithm. LNS was concluded to perform poorly compared to SA or TS when solving the emergency inspection disruption since minimizing the neighbourhood is inefficient in solving this type of disruption. It is also expected that this algorithm cant solve several emergency inspection disruptions due to the previous problem, although this was not verified in any of the tests.

Chapter 9

Conclusions and future work

9.1 Work synthesis

Previously to the implementation of the approach proposed by this dissertation, a profound revision to the state-of-the and also previous work done in the scope of this project were performed. In a preliminary phase, and to better understand DVRP optimization problems, several problem formulations in the literature were analysed. Problem's restrictions were also adapted from different DVRPs and incorporated into the approach here proposed.

After the problem understanding, and based on the state-of-the-art research, four optimization algorithms were selected: Hill-Climbing, Simulated Annealing, Tabu Search and Large Neighborhood search. These algorithms were proven to be valid and efficient in solving routing problems. Several improvements to the algorithms were explored and incorporated in each corresponding implementation to improve the performance and ability to navigate through the search space more intelligently.

Barros [27] explored and analysed the problem concerning this dissertation and proposed an approach to solving its static version. The static version of this problem doesn't consider disruptions and generates routes based on a utility function that maximises the amount of utility gathered by inspecting all the economic operators in the operational plan. Barros's approach was studied and used as an inspiration for this work. The following step was to get familiarized with the system developed in the context of project IA.SAE, especially the route generation module. All the system was studied, including the databases that provided the needed data to be used in this dissertation.

The OSRM routing app was selected based on state-of-the-art research by Barros on routing applications. This application was used to calculate the routes and the travel times between the several economic operators in the database system. This API's documentation was studied, and several functions were selected to be used in this approach. Alongside the routing app, the Leaflet, a framework capable of displaying the routes on a map, was also analysed and implemented into a simple web application.

A Disruption Generator module was created and implemented capable of generating six different disruption types to inspection routes and configurable using a set of defined parameters. After generated, the disruptions are added to the system, and the needed constraints are updated.

A module capable of solving problem instances was developed, receiving an initial solution with disruptions applied to it, outputting a new solution based on a utility function. This module comprises four different optimization algorithms: Hill-Climbing, Simulated Annealing, Tabu Search and Large Neighborhood search. These algorithms work exclusively, meaning that only the selected one is used to solve the problem instance.

The system developed in this dissertation includes a web application build to display relevant information about the routes and a map showing relevant sites' geographic position. This application receives data from ASAE's database and also from the Disruption Generator and optimisation modules. Everything is also connected with the routing app that provides the needed routing functionalities.

9.2 Conclusions and Results

The overall objectives were reached by analysing the work concerning this dissertation, culminating in implementing a system capable of generating disruptions to a set of vehicle routes and addressing them, always keeping the solution as optimized as possible. Although not capturing all the field constraints characteristic of ASAE's operational environment, this dissertation proposes a valid approach to tackle disruptions in a DVRP scenario. The disruptions and most of the constraints applied were inspired by ASAE's, meaning they are characteristic of systems where a vehicle fleet provides a service to a set of customers.

This dissertation created a module capable of generating disruptions over a set of inspection routes. These disruptions are generated using a set of parameters that control their intensity and their frequency.

This work proposed a complex utility function weighted and based on three different components, trying to capture the dynamic variant of the problem. The utility function depends on the utility gathered by visiting one route's economic operators, the similarity between the initial and the new solutions, and the average time each brigade gets to the depot at the end of the workday.

Another result of this dissertation was the comparison between four optimization algorithms used to solve similar instances of the problem. Overall, the algorithms provided reasonable solutions to almost the totality of the tests performed with few exceptions. The Hill-Climbing algorithm was concluded to have the fastest convergency, although it offers lower-quality solutions since it gets stuck in local maximums. This algorithm also fails to address the emergency inspection disruption. The Simulated Annealing offered a better solution quality when compared with HC, being able to solve all types of disruption. However, SA fails to deliver solutions with higher solution similarity. Returning solutions different from the initial ones will increase the operational costs, as explained in the literature review. The Tabu Search algorithm offered an outstanding balance between finding the best solution and in the shortest amount of time. TS also solved

all problem instances and returned solutions similar to the initial ones when the utility function benefits such behaviour. The Large Neighborhood Search algorithm was the one that consistently offered a better solution quality when compared to the others. Nevertheless, due to its implementation, this algorithm is sub-optimal compared to the SA and TS when solving the emergency inspection disruption.

9.3 Limitations

The approach developed simulated the generation of disruptions over operational plans and was not tested in a real-life scenario where the system is running, and disruptions happen in real-time. Several unexpected elements and possible weaknesses of this approach might be visible in the real-life scenario and influence its performance. Once a disruption happens, it is sent to the routing system, recalculating the whole operational plan. While the routing system is solving the optimization problem, all the brigades are in the terrain, either working or waiting for instructions. This dissertation didn't study the impacts of the computational time taken by the algorithms in the system. Therefore, this work can't make conclusions about the efficiency of a specific method in a real-life scenario.

This dissertation didn't study the scalability of the proposed approach. The solution was tested and reviewed over a problem instance with at most a dozen brigades and considered close to one thousand economic operators. The actual scenario of the problem has millions of economic operators located over the whole country. The possibility of a different routing system for each operation unit seems reasonable, but even that situation has a different level of magnitude than the one studied in this dissertation.

9.4 Future Development Perspectives

The approach used to generate the schedules, although based on real time tables, doesn't correspond to the real-life scenario. Future development could be implementing a way to gather the real schedules for all the economic operators in the system, either by using an existing API or by developing a method to collect the schedules in a semi-automatic way. The system implemented by this dissertation can adapt to the new schedules as long as they maintain the representation presented in section .

A further enhancement of the process can be obtained by dividing the solution optimisation once a disruptive event occurs in two processes. Firstly, an algorithm with less computational times (adjusting the tunable parameters) can be used to obtain a reasonably acceptable solution in a short time. This solution will be readily communicated to the vehicle fleet, and vehicles will start satisfying the new set of routes. After transmitting the solution, the system keeps optimising it by running more iterations and using more complex parameters. The solution will later be transmitted to the fleet in the case of new optimisations that may not have been accounted for in the primary solution. This approach would be similar to the one documented by Ritzinger. [44]

Some contexts may also benefit in using stochastic information, which is often provided by external elements such as the media or market trends. Useful stochastic information can also be provided by machine learning algorithms using data from the past. Stochastic information can be used to adapt further the initial solution to dynamic elements that are likely to happen during the execution of the planned routes. Methods such as Sampling 2.8.2.1, can be used to generate *dummy* instances of dynamic variables occurring in the problem's context.

Since this dissertation concerns a problem instance in a dynamic environment, other information could be incorporated to represent the real-life scenario more accurately and provide additional route rearrangement tools. Information like traffic patterns at a particular time of the day or on specific days of the week or access could provide a more accurate calculation of the travel times in the route network. Weather information could also be incorporated, reducing the average speed of the vehicles in the system in adverse meteorologic conditions.

This dissertation used a fixed inspection time of one hour if no disruption happens. In a real-life scenario, inspection times will depend on several factors: the type of economic operator inspected, its size, or other macro and micro-factors. This information could be used to forecast an inspection time closer to reality, increasing this approach's efficiency in a real-life scenario.

Appendix A

Economic Operators

A.1 Economic Operators distribution

Location	Quantity	Location	Quantity
CARVALHAIS - MIRANDELA	1	VILA NOVA DE FOZ CÔA	6
BRAGANCA	2	COTAS	1
LAMEGO	41	VILARINHO DOS FREIRES	1
VILA REAL	73	VILAR DE MAÇADA	1
VILA NOVA DE FOZ COA	2	REBORDAINHOS	1
GOSTEI	1	SESULFE	1
SÃO TOMÉ DO CASTELO	2	CONSTANTIM VRL	1
VILA CHÃ DE BRACIOSA	1	SANTA CRUZ-TRINDADE	1
CHAVES	46	VINHAIS	6
MIRANDELA	33	TORGUEDA (VILA REAL)	1
SANTA MARIA MAIOR	1	FAILDE	1
VIMIOSO	3	MADALENA	1
VILA FLOR	11	CARVIÇAIS	1
CARRAZEDA DE ANSIÃES	8	CARRAGOSA	1
ARMAMAR	13	TORRE DE MONCORVO	3
MURÇA	5	SABROSO DE AGUIAR	1
NUMÃO	1	LALIM	1
PESO DA RÉGUA	14	CARVA	1
SENDIM	4	VENDA NOVA	1
MACEDO DE CAVALEIROS	15	CARRAZEDA DE ANSIAES	1
VALPAÇOS	15	FREIXO ESPADA A CINTA	3
ALIJO	7	SAO JOAO DA PESQUEIRA	1
BRAGANÇA (SÉ)	1	TORRE DONA CHAMA	1
MOGADOURO	8	TABUACO	2

BRAGANÇA	41	MONCORVO	2
PESO DA REGUA	6	VILARINHO DE AGROCHÃO	1
ALIJÓ	8	STA MARTA DE PENAGUIAO	1
SÃO JOÃO DA PESQUEIRA	5	VILARES	1
ALFÂNDEGA DA FÉ	1	FREIXO DE ESPADA À CINTA	1
CONSTANTIM	2	VIADE DE BAIXO	2
MOURA MORTA PRG	1	LAMEGO (SÉ)	1
PINHÃO	2	ANDRÃES	2
LODÕES	1	FERREIRIM LMG	1
ALFÂNDEGA DA FÉ	5	REBORDELO	1
GODIM	4	PAREDES DA BEIRA	2
MONTALEGRE E PADROSO	1	PENEDONO	1
VILA NOVA DAS PATAS	1	CARRAZEDO DE MONTENEGRO	2
TABUAÇO	4	CEDÃES	1
TAROUCA	11	MALHADAS	1
ADOUFE	2	VALE DA PORCA	1
MONTALEGRE	4	S. MAMEDE DE RIBATUA	1
SANTA COMBA DE VILARIÇA	1	LOUREIRO	1
SALTO	2	MOUÇÓS	1
CAÇARELHOS	1	FOLHADELA - VILA REAL	1
CANDEDO	1	MONDIM DA BEIRA	2
SABROSA	6	VALDANTA	1
ARGOZELO	1	MESÃO FRIO	2
SANTA MARTA DE PENAGUIÃO	1	MASCARENHAS	1
VILA POUCA DE AGUIAR	6	FONTES	1
MORAIS	1	GIMONDE	2
TORRE DE DONA CHAMA	2	VILARINHO DA RAIA	1
BOTICAS	2	REBORDÃOS	1
FAVAIOS	2	VIDAGO	1
ERVEDOSA DO DOURO	2	GONDESENDE	1
CASTELÃOS - MACEDO DE CAVALEIR	1	FELGAR - TORRE DE MONCORVO	1
MIRANDA DO DOURO	7	FORNOS	1
VALE DE PRADOS	2	BRANGANÇA	1
SÃO MARTINHO DE ANTAS	2	TELÕES	1
PADORNELOS	2	IZEDA	1
TÓ	1		

Appendix B

Algorithms Code

```
    for turn in schedule:
        if (current_time < turn[0]): #waits to open
            return turn[0], 0, 0
        elif((turn[0] <= current_time) and (turn[1] >= current_time)):
            return current_time, 0, 0
    ###Failed Inspection - schedule not available#####
     return False, 0, 0
```

Listing B.1: Function used to check a soft schedule. Returns the start time of an inspection or False if the economic operator can't be inspected.

```
    if HARD_SCHEDULE_CONSTRAINT:
        for turn in schedule:
            temp = current_time + inspection_time
            if((turn[0] <= temp) and (turn[1] >= temp)):
                if current_time >= turn[0]:#the inspection can start now
                    return current_time, 0, 0
            elif(current_time < turn[0]):
                temp = turn[0] + inspection_time
                if(temp <= turn[1]): #inspects in the first available slot (wait)
                    return turn[0], 0, 0
        ###Failed Inspection - schedule not available#####
        return False, 0, 0
```

Listing B.2: Function used to check a hard schedule. Returns the start time of an inspection or False if the economic operator can't be inspected.

```
def algorithm(inspection_times, operators, indexes, travel_times, initial_solution,
     starting_time, starting_date, utility_function, emergency_operators, debug =
    False): #unfeasible, allows the use of unfeasible solutions
```

```
    graph_x_values = []
    graph_y_values = []
    weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
        'Sunday']
    print(weekdays[starting_date.weekday()])
    weekday = weekdays[starting_date.weekday()]
    i = 0
    best_solution_utility = 0
    best_solution = []
    best_solution = initial_solution
    best_solution_utility, last_time, penalty = algorithms_functions.
        solution_utility(inspection_times, initial_solution, best_solution, indexes
        , travel_times, starting_time, weekday, utility_function,
        emergency_operators) #calculates a solution's utility
    if penality > 0:
        best_solution_utility = best_solution_utility-penality
    while(i<=N_STEPS):
        new_solution = algorithms_functions.generateNewSolution(best_solution,
            operators)#generates a new solution
        new_solution_utility, last_time, penalty = algorithms_functions.
            solution_utility(inspection_times, initial_solution, new_solution,
            indexes, travel_times, starting_time, weekday, utility_function,
            emergency_operators)
        if penality > 0:
            new_solution_utility = new_solution_utility - (algorithms_functions.
                func_C(i) * penality)

        if(new_solution_utility >= best_solution_utility):
            best_solution_utility = new_solution_utility
            best_solution = new_solution
        i += 1

    return best_solution_utility, best_solution
```

Listing B.3: Hill-Climb Algorithm

```
    weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
        'Sunday']
    weekday = weekdays[starting_date.weekday()]
    best_solution_utility = 0
    best_solution = []
    reheat = 0 #reheat counter
    p_const = np.log(np.log(TEMP_O/TEMP_F)/np.log(A))
    q_const = np.log(1/F)
    b_const = p_const/q_const
    current_solution = initial_solution
    cur_solution_utility, last_time, penality = algorithms_functions.
        solution_utility(inspection_times, initial_solution, current_solution,
```

```
        indexes, travel_times, starting_time, weekday, utility_function,
        emergency_operators)
11      cur_solution_utility = cur_solution_utility - (algorithms_functions.func_C(1) *
         penality)
12      temp = calculateInitialTemp(inspection_times, initial_solution, operators,
        indexes, travel_times, starting_time, weekday, utility_function,
        emergency_operators) #Calculates the initial temperature
13      stable_temperature_treshold = BASE_TEMPERATURE_PERCENTAGE * N_ITE_MAX  #No
        Iterations Initial Temperature
14      final_temperature_treshold = N_ITE_MAX - BASE_TEMPERATURE_PERCENTAGE *
        N_ITE_MAX  #No Iterations null Temperature
15      initial_solution_utility = cur_solution_utility
16      while reheat < REHEAT_MAX: #Reheat
17          current_solution = initial_solution
18          i = 0
19          cur_solution_utility = initial_solution_utility
20          markov_chain_number = 0
21          chains_without_update = 0 #Markov chains without update
22          while ((i < N_ITE_MAX or chains_without_update <
                MARKOV_CHAIN_WITHOUT_IMPROVEMENT_TRESHOLD) and markov_chain_number<
                K_MAX): #Stopping criteria
23              new_solution, new_solution_utility = algorithms_functions.
                    generateSolutions(inspection_times, initial_solution,
                    current_solution, operators, indexes, travel_times, starting_time,
                    weekday, i, utility_function, emergency_operators)
24              utility_diff = new_solution_utility-cur_solution_utility
25              prob = min (utility_diff/temp, 700)
26              if (random.random() <= math.exp(prob)): #acceptance condition
27                  current_solution = new_solution
28                  cur_solution_utility = new_solution_utility
29                  if i >= N_ITE_MAX:
30                      if utility_diff > MIN_IMPROVEMENT_CONSIDERED:
31                          chains_without_update = 0
32                      else:
33                          chains_without_update += 1
34                  markov_chain_number += 1
35                  #########Cooling schedule#############
36                  if i > stable_temperature_treshold:
37                      temp = TEMP_O * pow(A, -1*pow(markov_chain_number/(F*K_MAX),
                            b_const))
38                  elif i > final_temperature_treshold:
39                      temp = 0
40                  if cur_solution_utility > best_solution_utility: #Updates the best
                        solution
41                      best_solution_utility = cur_solution_utility
42                      best_solution = current_solution
43              i += 1
44          reheat += 1
45          temp = TEMP_O
```

```
46      return best_solution_utility, best_solution
```

Listing B.4: Simulated Annealing Algorithm

```
1       def generateSolutions(inspection_times, ...): #Generates 6 neighbours
2       best_solution = []
3       best_solution_utility = float('-inf')
4       best_penality = 0
5
6       for i in range(6):
7           new_solution = generateNewSolution(current_solution, operators, False, i) #
                generates 1 solution
8           if new_solution == False:
9               continue
10          new_solution_utility, last_time, penality = solution_utility(
                inspection_times, initial_solution, new_solution, indexes, travel_times
                , starting_time, weekday, utility_function, emergency_operators)
11          new_solution_utility = new_solution_utility - (func_C(iter) * penality) #
                apply penality when violating contraints
12          if new_solution_utility > best_solution_utility:
13              best_solution_utility =  new_solution_utility
14              best_solution = new_solution
15      return best_solution, best_solution_utility
```

Listing B.5: Function used to select the best neighbours from 6 solutions in the neighbourhood

```
1   for i in range(neighbour_number):
2       selected_operation = i % different_operations #Operator index
3       new_solution, new_operators, ... = algorithms_functions.generateNewSolutionTabu
            (...) #generates a new solution
4       if new_solution == False: #operator failed to generate solution
5           continue
6       new_solution_utility, last_time, penality = algorithms_functions.
            solution_utility(...) #calculate solution utility
7       new_solution_utility = new_solution_utility - penality #apply penality when
            violating contraints
```

Listing B.6: Tabu-Search Neighbourhood generation

```
1   if (not was_tabu) or (new_solution_utility > best_solution_utility) or (
        new_solution_utility > old_tabu_solution_utility): #checks if a move should be
        accepted
2       if new_solution_utility > best_neighbor_utility: #checks if a solution is the
            best found so far
3           best_neighbor_utility =  new_solution_utility
```

```
4           best_neighbor = new_solution
5           best_operators = new_operators.copy()
6           best_random_operator = random_operation
7           best_penality = penality
```

Listing B.7: Tabu move verification and use of the Aspiration criteria

References

[1] OpenStreetMap. Available at https://www.openstreetmap.org/, Accessed last time in August, 2021.

[2] Vladimir Agafonkin. Leaflet — an open-source JavaScript library for interactive maps. Available at http://www.leafletjs.com/,version 1.7.1, Accessed last time in August, 2021.

[3] Claudia Archetti, Francesca Guerriero, and Giusy Macrina. The online vehicle routing problem with occasional drivers. *Computers and Operations Research*, 127:105144, 2021.

[4] Mir M. Atiqullah. An efficient simple cooling schedule for simulated annealing. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3045:396–404, 2004.

[5] T. Barros, A. Oliveira, H.L. Cardoso, L.P. Reis, C. Caldeira, and J.P. Machado. Generation and optimization of inspection routes for economic and food safety. volume 2, pages 268–278, 2020.

[6] Telmo Barros, Tiago Santos, Alexandra Oliveira, Henrique Lopes Cardoso, Luís Reis, Cristina Caldeira, and João Machado. Interactive inspection routes application for economic and food safety. pages 640–649. 05 2020.

[7] Dimitris Bertsimas, Patrick Jaillet, and Sébastien Martin. Online vehicle routing: The edge of optimization in large-scale applications. *Operations Research*, 67(1):143–162, 2019.

[8] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, 2009.

[9] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35:268–308, 01 2001.

[10] Wan Tzu Chang, Yen Po Yeh, Hong Yi Wu, Yu Fen Lin, Thai Son Dinh, and Iebin Lian. An automated alarm system for food safety by using electronic invoices. *PLoS ONE*, 15(1):1–11, 2020.

[11] Huey Kuo Chen, Che Fu Hsueh, and Mei Shiang Chang. The real-time time-dependent vehicle routing problem. *Transportation Research Part E: Logistics and Transportation Review*, 42(5):383–408, 2006.

[12] Siyoung Chung, Mark Chong, Jie Sheng Chua, and Jin Cheon Na. Evolution of corporate reputation during an evolving controversy. *Journal of Communication Management*, 23(1):52–71, 2019.

[13] G. B. Dantzig and J. H. Ramser. The Truck Dispatching Problem. *Management Science*, 6(1):80–91, 1959.

[14] Autoridade de Segurança Alimentar e Económica. Plano de Atividades ASAE- 2021, November 2020.

[15] Richard Eglese and Sofoclis Zambirinis. Disruption management in vehicle routing and scheduling for road freight transport: a review. *Top*, 26(1):1–17, 2018.

[16] Wan Fang, Guo Haixiang, Li Jinling, Gu Mingyun, and Pan Wenwen. *Multi-objective Emergency Scheduling for Geological Disasters*, volume 105. Springer Netherlands, 2021.

[17] Wade Genders and Saiedeh N. Razavi. Impact of Connected Vehicle on Work Zone Network Safety through Dynamic Route Guidance. *Journal of Computing in Civil Engineering*, 30(2):04015020, 2016.

[18] Michel Gendreau, François Guertin, Jean-Yves Potvin, and Éric Taillard. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science*, 33:381–390, 11 1999.

[19] Fred Glover. Paths for Integer Programming. *Computers and Operations Research*, 13(5):533–549, 1986.

[20] Fred Glover, Eric Taillard, and Eric Taillard. A user's guide to tabu search. *Annals of Operations Research*, 41(1):1–28, 1993.

[21] Larry Goldstein and Michael Waterman. Neighborhood size in the simulated annealing algorithm. *American Journal of Mathematical and Management Sciences*, 8(3-4):389–407, 1988.

[22] Michael Hahsler and Kurt Hornik. TSP - Infrastructure for the traveling salesperson problem. *Journal of Statistical Software*, 23(2):1–21, 2007.

[23] Psaraftis Harilaos N. Dynamic vehicle routing: status and prospects. *Annals of Operations Research*, 61:143–164, 1995.

[24] Pascal Van Hentenryck and Russell Bent. *Online Stochastic Combinatorial Optimization*. The MIT Press, 2006.

[25] Soumia Ichoua, Michel Gendreau, and Jean-Yves Potvin. Planned route optimization for real-time vehicle routing. In *Dynamic Fleet Management*, chapter 1, pages 1–18. 2008.

[26] Cangyu Jin, Yamine Bouzembrak, Jiehong Zhou, Qiao Liang, Leonieke M. van den Bulk, Anand Gavai, Ningjing Liu, Lukas J. van den Heuvel, Wouter Hoenderdaal, and Hans J.P. Marvin. Big Data in food safety- A review. *Current Opinion in Food Science*, 36:24–32, 2020.

[27] Telmo João Vales Ferreira Barros. IA.SAE – Geração e Otimização das Rotas de Fiscalização. Master's thesis, Faculdade de Engenharia da Universidade do Porto, Mestrado Integrado em Engenharia Informática e Computação, July 2019. Supervisor: Luís Paulo Reis.

[28] Allan Larsen. *The dynamic vehicle routing problem*. PhD thesis, Technical University of Denmark, Lyngby, Denmark, December 2000.

[29] Jenn Long Liu and Jiann Horng Lin. Evolutionary computation of unconstrained and constrained problems using a novel momentum-type particle swarm optimization. *Engineering Optimization*, 39(3):287–305, 2007.

[30] K. Lund, Oli Madsen, and Rygaard J.M. Vehicle routing problems with varying degrees of dynamism. 01 1996.

[31] Dennis Luxen. Project open source routing machine (osrm). Available at http://project-osrm.org/, Accessed last time in June, 2021.

[32] Hans J.P. Marvin, Esmée M. Janssen, Yamine Bouzembrak, Peter J.M. Hendriksen, and Martijn Staats. Big data in food safety: An overview. *Critical Reviews in Food Science and Nutrition*, 57(11):2286–2295, 2017.

[33] I. Minis, K. Mamasis, and V. Zeimpekis. Real-time management of vehicle breakdowns in urban freight distribution. *Journal of Heuristics*, 18(3):375–400, 2012.

[34] Snežana Mitrović-Minić, Ramesh Krishnamurti, and Gilbert Laporte. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(8):669–685, 2004.

[35] R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati. Ant Colony System for a Dynamic Vehicle. *Journal of Combinatorial Optimization*, 10(4):327–343, 2005.

[36] Wasin Padungwech, Jonathan Thompson, and Rhyd Lewis. Effects of update frequencies in a dynamic capacitated arc routing problem. *Networks*, 76(4):522–538, 2020.

[37] Jing Pan, Min Huang, Qihuan Zhang, and Yang Yu. Dynamic Vehicle Routing Problem Considering Customer Satisfaction. *Chinese Control Conference, CCC*, 2020-July:5602–5606, 2020.

[38] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.

[39] David Pisinger and Stefan Ropke. Large neighborhood search. *International Series in Operations Research and Management Science*, 272(September):99–127, 2019.

[40] Warren B. Powell, Yosef Sheffi, Kenneth S. Nickerson, Kevin Butterbaugh, and Susan Atherton. Maximizing Profits for North American Van Lines' Truckload Division: A New Framework for Pricing and Operations. *Interfaces*, 18(1):21–41, 1988.

[41] Harilaos N. Psaraftis. Dynamic Programming Solution To the Single Vehicle Many-To-Many Immediate Request Dial-a-Ride Problem. *Transportation Science*, 14(2):130–154, 1980.

[42] Harilaos N. Psaraftis. Dynamic vehicle routing problems. In *Vehicle Routing: Methods and Studies*, pages 223–248. North-Holland, 1988.

[43] Harilaos N. Psaraftis, Min Wen, and Christos A. Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31, 2016.

[44] Ulrike Ritzinger, Jakob Puchinger, and Richard F. Hartl. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, 54(1):215–231, 2016.

[45] Armin Ronacher. Flask (2.0.x). Available at https://flask.palletsprojects.com/en/2.0.x/, Accessed last time in September, 2021.

[46] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[47] Bart Selman and Carla Gomes. Hill-climbing search. In *Encyclopedia of Cognitive Science*, pages 333–336. 01 2006.

[48] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1716, pages 417–431. 1999.

[49] Akshit Singh, Nagesh Shukla, and Nishikant Mishra. Social media data analytics to improve supply chain management in food industries. *Transportation Research Part E: Logistics and Transportation Review*, 114(June):398–415, 2018.

[50] DD SLEATOR and RE TARJAN. Amortized Efficiency of List. *Communications of the ACM*, 28(2):202–208, 1985.

[51] Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[52] Mariam Tagmouti, Michel Gendreau, and Jean Yves Potvin. A dynamic capacitated arc routing problem with time-dependent service costs. *Transportation Research Part C: Emerging Technologies*, 19(1):20–28, 2011.

[53] Eiichi Taniguchi and Hiroshi Shimamoto. Intelligent transportation system based dynamic vehicle routing and scheduling with variable travel times. *Transportation Research Part C: Emerging Technologies*, 12(3-4 SPEC.ISS.):235–250, 2004.

[54] Niaz A. Wassan, A. Hameed Wassan, and Gábor Nagy. A reactive tabu search algorithm for the vehicle routing problem with simultaneous pickups and deliveries. *Journal of Combinatorial Optimization*, 15(4):368–386, 2008.

[55] N.H.M. Wilson and N.J. Colvin. *Computer Control of the Rochester Dial-A-Ride System*. CTS report. Massachusetts Institute of Technology, Center for Transportation Studies, 1977.

[56] Haitao Xu, Pan Pu, and Feng Duan. Dynamic Vehicle Routing Problems with Enhanced Ant Colony Optimization. *Discrete Dynamics in Nature and Society*, 2018:1–13, 2018.

[57] Xin Yao. Dynamic Neighbourhood Size in Simulated Annealing. *Proc. of Int'l Joint Conf. on Neural Networks (IJCNN'92)*, pages 1–7, 1992.

[58] Gang Yu and Xiangtong Qi. Disruption management: Framework, models and applications. *World Scientific*, 08 2004.