FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Genetic Programming approaches for solving transportation problems

Catarina Furtado Martins da Rocha Leite

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Luís Gonçalo Rodrigues Reis Figueira, PhD

Co-Supervisor: Fábio Neves Seabra da Silva Moreira, PhD

October 3, 2022

Abstract

Over the last decade, Artificial Intelligence (AI) has experienced significant growth in society with remarkable performance in many domains. However, despite the high-level performance of existing models, their lack of explainability has challenged humans to understand them. Therefore, there is a need to produce equally performant yet explainable models. Symbolic learning methods have been used to assist in the achievement of this trade-off between performance and interpretability.

Genetic Programming (GP) is a symbolic learning method used extensively in different applications, from regression to optimisation problems, especially dynamic ones, where information is revealed gradually, and decisions need to be made along the time. In these settings, decisions might need to be made in real-time, thus preventing the use of exact optimisation methods or even meta-heuristics, and pointing to extremely fast solution methods, such as simple heuristics or dispatching rules. These heuristics can be devised manually or automatically (with hyper-heuristics). The latter is where GP can excel due to its flexible representation and high-level explainability. The recent advances in data collection technologies (e.g. IoT, Digital Twins, 5G) are fostering the development of new approaches to solve dynamic problems, such as the dynamic Vehicle Routing Problem (VRP). These approaches need to be extremely efficient so that accurate transportation costs can be determined in fractions of a second.

The present work applies GP techniques to solve dynamic VRP instances related to home attended deliveries in the online retail business sector. Very few works have been found in the literature regarding the use of GP methods to solve VRPs, encouraging the investigation of the topic. We start by analysing the behaviour of different GP algorithms in a Traveling Salesman Problem simulator with delivery time windows. Next, the selected GP algorithm is used to generate routing policies that solve dynamic VRPs with delivery time windows. In order to do that, transportation problem-related features and hyper-parameter combinations are first studied and tested. Several routing policies are generated and can be chosen by trading-off their performance and size. A case study in a large European online retailer is then analysed, and the real-world data is collected and used to plan the routes of the retailer with the GP approach. Additionally, a set of well-known heuristics is designed to be used in the comparison of the results.

Using real-world data, the proposed GP approach can improve dynamic VRP solutions while increasing the interpretability and the responsiveness of real-time decisions. By applying the routing policies evolved by GP, the retailer routes cost is reduced by almost 50% and the number of used vehicles by more than 40%. Furthermore, the results suggest that routing policies evolved by GP can outperform manually-designed heuristics that are commonly used in practice.

Keywords: Genetic Programming, Vehicle Routing, Explainability, Online Retail

Resumo

Durante a última década, a Inteligência Artificial (AI) conheceu um crescimento significativo na sociedade com um desempenho notável em muitos domínios. No entanto, apesar do desempenho de alto nível dos modelos existentes, a sua falta de explicabilidade tem desafiado os humanos a compreendê-los. Por conseguinte, há necessidade de produzir modelos com igual performance mas explicáveis. Métodos de aprendizagem simbólicos têm sido utilizados para ajudar a alcançar este compromisso entre desempenho e interpretabilidade.

A Programação Genética (GP) é um método de aprendizagem simbólica amplamente utilizado em diferentes aplicações, desde a regressão a problemas de optimização, especialmente os dinâmicos, onde a informação é revelada gradualmente, e as decisões têm de ser tomadas ao longo do tempo. Nestas situações, decisões podem ter de ser tomadas em tempo real, impedindo assim a utilização de métodos exactos de optimização ou mesmo meta-heurísticas, e apontando para métodos de solução extremamente rápidos, tais como heurísticas simples ou regras de expedição. Estas heurísticas podem ser concebidas manual ou automaticamente (com hiper-heurísticas). Esta última é onde o GP pode destacar-se devido à sua representação flexível e explicabilidade de alto nível. Os recentes avanços nas tecnologias de recolha de dados (e.g. IoT, *Digital Twins*, 5G) estão a promover o desenvolvimento de novas abordagens para resolver problemas dinâmicos, tais como o problema dinâmico de roteamento de veículos (VRP). Estas abordagens precisam de ser extremamente eficientes para que custos de transporte precisos possam ser determinados em fracções de segundo.

O presente trabalho aplica técnicas de GP para resolver casos dinâmicos de VRP relacionados com entregas ao domicílio no sector do comércio de retalho *online*. Muito poucos trabalhos foram encontrados na literatura sobre a utilização de métodos de GP para resolver VRPs, encorajando a investigação do tema. Começamos por analisar o comportamento de diferentes algoritmos de GP num simulador de Problemas do caixeiro-viajante com janelas de entrega. Em seguida, o algoritmo de GP seleccionado é utilizado para gerar políticas de roteamento que resolvem VRPs dinâmicos com janelas de entrega. Para o efeito, são estudadas e testadas características relacionadas com o problema e combinações de hiper-parametros. Várias políticas de roteamento são geradas e escolhidas através da relação entre o seu desempenho e tamanho. Um caso de estudo num grande retalhista *online* Europeu é então analisado, e os dados do mundo real são recolhidos e utilizados para planear as rotas do retalhista com a abordagem do GP. Além disso, um conjunto de heurísticas é concebido para ser utilizado na comparação dos resultados.

Utilizando dados do mundo real, a abordagem de GP proposta melhora as soluções VRP dinâmicas ao mesmo tempo que aumenta a interpretabilidade e a capacidade de resposta das decisões em tempo real. Ao aplicar as políticas desenvolvidas pelo GP, o custo das rotas do retalhista é reduzido em quase 50% e o número de veículos usados em mais de 40%. Além disso, os resultados sugerem que as políticas de roteamento desenvolvidas pelo GP têm um desempenho superior às heurísticas de concepção manual que são normalmente utilizadas na prática.

Palavras-chave: Programação Genética, Roteamento de Veículos, Explicabilidade, Retalho Online

Acknowledgements

To my supervisors, Prof. Gonçalo Figueira and Prof. Fábio Neves Moreira, for all the support during this period. It has been a challenging and exciting semester in which I got out of my comfort zone, learned many new concepts and grew a lot. This would not be possible without all the guidance and dedication they provided.

To Prof. Pedro Rocha and Prof. Cristiane Ferreira for their contributions to the development of this work.

To the CEGI team, these past few months would not have been the same without the moments at lunchtime at INESC TEC with an extremely welcoming team that provided enjoyable break times and gave me motivation for the development of the work. I am incredibly grateful for these moments at CEGI.

To my parents, I would like to thank them for all the opportunities they made possible and for their unconditional support.

To all my family, for all the love and joy with which they celebrate my achievements.

To my friends, who were a vital part of this academic path.

And finally, to João for the constant support, care and joy.

"Live as if you were to die tomorrow.
Learn as if you were to live forever."

Mahatma Gandhi

Contents

List of Figures

List of Tables

Abbreviations and Symbols

AI	Artificial Intelligence
AVGT	Average Time from remaining nonserved customers to the considered customer
AVGTL	Average Time from remaining nonserved customers to the Last customer added
C	Extra Cost of serving a customer
CIH	Cheapest Insertion Heuristic
CR	Critical Ratio
CVRP	Capacitated Vehicle Routing Problem
DAGP	Dimensionally Aware Genetic Programming
DVRPTW	Dynamic Vehicle Routing Problem with Time Windows
ETW	End of the Time Window
GA	Genetic Algorithms
GGGP	Grammar-Guided Genetic Programming
GP	Genetic Programming
GPHH	GP-based Hyper-Heuristic
GOMEA	Gene-pool Optimal Mixing Evolutionary Algorithm
HFVRP	Heterogeneous Fleet Vehicle Routing Problem
NNA	Nearest Neighbour Algorithm
OR	Operations Research
Q	Demand
RC	Remain Capacity
RHH	Ramped Half-and-Half
RL	Reinforcement learning
RT	Return Time to the depot
RTT	Route Total Time
NRTT	New Route Total Time
S	Slack time
ST	Service Time
STGP	Strongly-Typed Genetic Programming
STW	Start of the Time Window
TSP	Traveling Salesman Problem
TSPTW	Traveling Salesman Problem with Time Windows
U	Utilization
VRP	Vehicle Routing Problem
VRPDP	Vehicle Routing Problem with Pick-up and Delivery
VRPTW	Vehicle Routing Problem with Time Windows
XAI	Explainable Artificial Intelligence

Chapter 1

Introduction

Artificial Intelligence (AI) is having an increasingly positive impact on society, with successful applications in many domains. However, as AI becomes more advanced, humans are challenged to understand the existing models, such as deep neural networks, due to their black-box nature. Therefore, there is the need to produce explainable models while maintaining their effectiveness, enabling humans to trust them [1].

Accordingly, the concept of Explainable AI (XAI) emerged by attempting to offer an acceptable trade-off between explainability and performance [2]. However, most XAI approaches are focused on building local explanators on top of robust black-box models instead of proposing explainable-by-design or white-box models [3]. Therefore, despite the existence of post-hoc models (e.g. Lime [4] and SHAP [5]) that explain the outcome of black-box models, there is the need to learn models that are interpretable by their structure. This is the purpose of the European project TRUST-AI, which promotes a new human-guided learning paradigm to evolve symbolic learning models that humans can understand and interpret. The ultimate goal of this project would be to develop models that are as effective as AI black-boxes and as explainable as theory-based approaches.

Although that might not be feasible in some applications, in others it might be. Combinatorial optimisation problems, for instance, have an underlying structure that can be learned, and symbolic models might be able to represent them. Previous studies have revealed promising results in multiple fields, such as scheduling [6], inventory management [7, 8] and physics [9]. The models developed within the scope of the project will be validated in several case studies in different fields. In the context of one of the case studies, in the online retail business sector, a new time slot management process intends to be implemented, which demands quick transportation cost approximations.

This dissertation will focus on the necessity faced by the case study of quickly estimating the transportation costs associated with a new client that just joined the online platform and is making a purchase. In order to do that, Genetic Programming (GP) approaches will be explored and employed to generate mathematical expressions that can be used to solve real-world transportation problems.

GP is a symbolic-based evolutionary learning approach with a great potential to contribute to XAI since it is a white-box approach, tending to be interpretable [3]. It produces a symbolic expression that can be more easily interpretable due to its tree representation and high transparency in constructing solutions. One should note that these symbolic expressions have a huge potential since they can be used to solve predictive and prescriptive problems in different business sectors.

GP, inspired by biological evolution, breeds a generation of computer programs. Through an iterative process, a new population of programs is generated by applying Darwin's natural selection and the effects of operators analogous to natural genetic processes (e.g. crossover, mutation) [10]. This procedure is repeated until a termination criterion is satisfied (e.g. the maximum number of generations reached). The result is a mathematical expression that can be saved and used to solve different instances of a problem in fractions of a second. GP is seen as an extension of Genetic Algorithms (GA) that allows the evolution of computer programs. The main difference between GP and GA is that in GP, the solution is represented by a computer program, while in GA, it is a fixed-length string of characters. Due to its representation power, GP can handle problems that are harder for GA to manage.

Over the years, GP has produced promising results in various types of problems in very different contexts such as robotics, game playing, analogue electrical circuits, and image recognition [11]. Also, in the field of Operations Research (OR), many problems have been solved using this technique due to its flexible representation and high-level explainability. One of the most challenging problems in OR is the Vehicle Routing Problem (VRP). This problem is widely studied in the area, appearing in many dynamic real-world problems such as the delivery time slot pricing [12] and the meal delivery problem [13]. Therefore, there is a necessity to produce equally efficient but more easily interpretable solutions in a shorter period of time for VRPs.

The VRP is a very challenging combinatorial optimisation problem typically solved using heuristics [14–16] or meta-heuristics [17–19]. In contexts where real-time transportation decisions need to be made, as new customers' requests arrive, these methods fail to provide solutions in a reasonable time since they are founded on an iterative search paradigm, which typically requires long times to produce high-quality solutions. Accordingly, there is a need for new approaches that, in fractions of a second, can solve VRPs. GP approaches, apart from being able to generate interpretable-by-design models, its evolved mathematical expressions can, very quickly, construct solutions to a problem.

1.1 Motivation

AI is an emergent field in several areas, but the existing approaches are virtually impossible to interpret by humans. Therefore, there is a need to produce equally explainable and efficient methods. Thus, GP has been used in the literature to assist in the interpretation of complex machine learning models, such as in [20, 21].

Additionally, constant advances in data collection technologies and the growing adoption of the just-in-time paradigm are triggering competition among companies in the transportation sector

and increasing customers' expectations for service quality. Therefore, the development of more efficient transportation planning systems is a necessity. Problems related to finding optimal routes in a distribution system are generally considered VRPs. A growing number of researchers are studying these problems since they are one of the most serious challenges faced by the logistic industry.

Since VRPs are NP-hard problems [22], they are computationally expensive to solve. Also, the incorporation of real-world data involves complex operational constraints such as time windows, variable travel times (e.g. due to traffic congestion and vehicle breakdowns), multiple depots, and heterogeneous fleets [22]. GP has been successfully applied to several optimisation problems and can be used to solve VRPs. Moreover, scarce works were found in the literature regarding this subject, encouraging the investigation of the topic.

The developed solution's approach will be validated in a real-world transportation problem faced by a large European online retailer. In the context faced by the retailer of the considered case study, it is critical to instantly estimate the transport costs associated with a new client that just joined the online platform. By accomplishing the desired goals of this dissertation, it will be possible to find a mathematical expression capable of satisfying this necessity.

The motivation for approaching VRPs with GP is that, as shown in [23] and [24], heuristics evolved by GP produced significantly better results when compared to manually designed heuristics, which are typically used in practice. Additionally, the model's interpretability has been improved when compared to black-box models. These results seem promising and encourage the application of GP approaches to complex, real-world problems.

1.2 Objectives

This dissertation aims to develop a GP approach that supports human-guided XAI and apply it to complex transportation problems modelled as VRPs.

A simulation model for a real-time VRP with delivery time windows is developed. This is a very complex transportation problem which considers constraints regarding vehicles' capacity and delivery time windows and, simultaneously, the need to reconstruct the routes as new customer requests arrive in real-time. The main goal is to find problem solutions that minimise the total cost of the routes, including the total travelled distance cost and the cost of postponed customer requests. Simultaneously, late arrivals at the customers must be avoided.

In the preparatory phase of the development of the solution, there is a need to test and investigate which options regarding certain aspects (e.g. GP frameworks, transportation-related features, hyper-parameter combinations) are the most adequate for the present scenario. First, GP algorithms available in the literature are studied, and their performance in simple problem instances is compared to inquire which approach suits best in the present problem. Next, relevant problem features are extracted, and their importance in the evolutionary process is examined. Simultaneously, a tunning procedure is performed to find an adequate hyper-parameter combination. Afterwards, by spreading the conceptualised model, the solutions are obtained.

Specifically, this dissertation aims at answering three main research questions:

- **RQ1** - What are the most important features to consider when developing a real-world routing problem solution?
- **RQ2** - Can genetic programming approaches find (near-)optimal solutions for the dynamic vehicle routing problem with time windows?
- **RQ3** - What is the trade-off between explainability and performance when employing GP-based approaches?

In order to answer these questions, a number of strategies should be formulated and adopted. Regarding the first question, discovering the group of most adequate features for the problem we aim to solve involves a deep knowledge of the problem. After that, an iterative process of testing and examining different features starts to inquire which ones are the most important to consider for generating effective solutions for the given problem. The definition of the problem characteristics used as features is critical for the algorithm's effectiveness, since they will dictate which variables are weighed in the routing policies achieved. Additionally, it is essential to ensure that all features included in the set are helpful for the process since the use of irrelevant features enlarges the algorithm search space, making it harder to achieve promising regions.

Concerning the second research question, this process will involve the use of generated instances of small size to inquire if the algorithm is able to find optimal solutions for the problem or if it gets stuck in local optima. If it is the case of the latter, techniques such as increasing mutation rate or redesigning genetic operators can be adopted.

The process of answering the last question will entangle the analysis of the solutions found by simultaneously evaluating their interpretability and performance. The level of interpretability of an expression is directly related to its size, so by limiting the number of nodes an expression can have, its size can be restricted. Also, the number of features included in the construction of the solutions, as well as their intuitiveness, can affect the ease of comprehensibility.

1.3 Dissertation Structure

The remainder of this document is organised as follows: Chapter 2 provides an overview of the theoretical background and describes the literature review from the existing studies on the subjects covered in this dissertation, namely, GP and VRPs. Chapter 3 details all relevant information regarding the problem being solved, starting with its formal definition, followed by the proposed solution approach. Chapter 4 presents the general experiment design, the obtained routing policies, and its results with instances from the literature. This chapter also describes the procedure to select the GP algorithm employed. The case study is then introduced in Chapter 5. Its operational necessities are presented, and the process of constructing the real instances is described. Then, the solutions obtained with the model are exposed as well as a comparison of the results achieved with the results of manually designed heuristics and the routes performed by the company in the

same planning horizon. Finally, Chapter 6 concludes the document, summarizing contributions of this work, conclusions and directions for future research.

Chapter 2

Literature Review

This chapter aims to address the fundamental concepts and existing studies in the main subjects covered in the development of this dissertation. It starts with an overview of the GP concept, its main characteristics and the introduction of some GP variants (Section 2.1). Then, it focuses on the theoretical aspects of VRPs, followed by the enumeration of the most common approaches to solving these types of problems (Section 2.2). Additionally, to highlight the potential of GP, its application in other combinatorial problems is described (Section 2.3). Finally, existing solutions for VRPs using GP are listed and evaluated for their adaptability to our project (Section 2.4).

2.1 Genetic Programming - GP

Genetic Programming has been used widely in different combinatorial optimisation problems such as production scheduling [6], arc routing problem [25], and knapsack problem [26].

The concept of GP was initially presented in [10], where Koza tries to answer the arising question: "can complex computer programs be created, not via human intelligence, but by applying a fitness measure appropriate to the problem environment?" [10, 87].

Individuals in the GP population are usually represented as symbolic expression trees and correspond to hierarchical combinations of primitive functions and terminals, defined in a preparatory phase by the human and adequate to the problem at hand. The set of primitive functions, which are internal nodes of the tree, is typically composed of mathematical functions and arithmetic, boolean, conditional and iterative operations. The terminals are leaves of the tree and include the variables and constants in the program. One should note that two requirements must be satisfied while specifying the set of functions and terminals: closure and sufficiency.

The closure property ensures that the programs are executable, implying that each function is defined for any possible combination of arguments. Respecting the closure property may not be simple, particularly when considering real-world applications since it may require different data types. In such cases, employing a Strongly-typed GP can help enforce it [27].

The sufficiency property forces the set of primitive functions and the set of terminals to express an adequate solution to the problem. The accomplishment of this property highly depends on the GP developer [27].

The genetic process starts with a population of mathematical expressions randomly created with different sizes and shapes. Then, a value (known as fitness) is assigned to each individual based on its ability to solve the problem of interest. The ones with the best performance pass directly to the next generation. From the surviving individuals, new expressions are generated through crossover and mutation. The above steps are applied, over many generations, to the population of mathematical expressions, which tends to become iteratively better.

The tree-based representation of GP solutions, as well as the transparency in the solutions' construction process, contribute to the high interpretability of this approach. Also, the expert knowledge required to define the set of functions and terminals permits the incorporation of domain specific information in the evolutionary process. Furthermore, in contrast to other methods, such as GA, where fixed-length individuals are evolved, GP trees have variable length representation, which is helpful if the best size for solutions is unknown. All these topics represent advantages of the employment of GP approaches to solve a variety of engineering problems.

2.1.1 Genetic Programming parameters

Vanneschi and Poli [27] listed the most relevant parameters in GP: population size, termination criterion, the procedure used to generate initial random population, selection technique, crossover and mutation type and rate, maximum tree depth, steady-state or generational update strategy, use or not of automatic defined functions and the presence or not of elitism.

The configuration of genetic parameters denotes a critical choice for algorithm performance, and care must be taken to choose them. Two main techniques are used to seek the best parameter configuration: tunning and control [28]. Tunning is the process of determining suitable values for the parameters before running the algorithm and then executing it with these values, which remain constant during the run. On the other hand, a control strategy for seeking parameters consists of changing the parameters during the run and adapting them to improve the evolved rules. Pedroso [29] presents a strategy to automatically adapt the parameters of an evolutionary algorithm to any optimisation problem of its class. This strategy consists of randomly changing them whenever a new algorithm generation does not produce improvements in the solution.

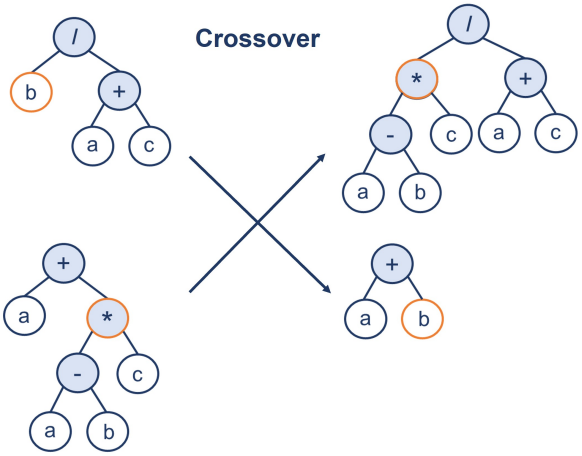
Regarding the methods used to initialise a GP population, the most commonly used methods are the grow, the flow, and the ramped half-and-half method [10]. The most widely used is the latter, which results from combining the other two methods. The ramped half-and-half method employs a range of tree depth limits to produce trees of different sizes and shapes.

Fitness values are used to probabilistically select individuals in the GP population. Different measures exist to calculate this value. Raw fitness is the most straightforward measurement, and consists of the sum of the differences between expected values from an ideal solution and the values actually returned. Standardised fitness is a modification of the previous measure and intends to improve lower numerical values. Regarding the selection of individuals based on their fitness to

participate in the genetic operations, the most commonly employed method is tournament selection. This approach starts by randomly picking a number of individuals from the population and then, after comparing, returns the best one of them [27].

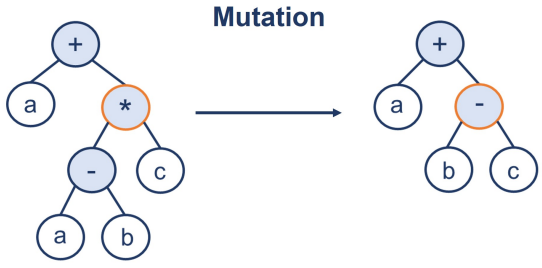
The crossover operator creates variation in the population by recombining randomly selected parts of two parents and creating two new offspring programs. Subtree crossover is the name of this method and is the most common crossover type used. Koza [10] suggested the use of a selection probability for crossover points of 0.9 for functions and 0.1 for terminals. This fact aims to guarantee that both parents contribute equally to their offspring's genetic code.

In the mutation operator, a new offspring program is created by first choosing a mutation point at random in a GP individual. Then, the subtree rooted in the mutation point is replaced with a new randomly generated subtree. This approach is called subtree mutation and is the most commonly used [27]. Figure 2.1 and 2.2 exemplify a GP crossover and a GP mutation operation, respectively.



The crossover point is represented by an orange circle.

Figure 2.1: Crossover operator in GP.



The mutation point is represented by an orange circle.

Figure 2.2: Mutation operator in GP.

While crossover is commonly seen as the operator responsible for fitness gains, the mutation is known for its ability to create variation in the population by reintroducing lost individuals back into the population or by creating never seen ones. Although traditionally, the crossover is known for being a more beneficial addition to the algorithm, studies have shown that no significant differences in the results exist when varying both operators' rates [30, 31].

2.1.2 Genetic Programming limitations

Although standard GP has been successfully applied to diverse problems, some issues related to domain space's size, modularity and constraint can occur, compromising its effectiveness. Accordingly, studies have risen in exploring GP variants that try to minimise these issues. For instance, Dimensionally Aware GP (DAGP) tries to reduce the solution space of GP, enhancing its interpretability [32], Strongly-Typed GP (STGP) applies data type constraints [33, 34], and Grammar-Guided GP (GGGP) enforces dimensionally consistent expressions by using a grammar [34].

GP variants are typically more powerful but also more challenging to implement, mainly because genetic operators are significantly more complex. Although, in some applications, it is necessary to construct a more robust structure of GP solutions. Ratle and Sebag [35] proposed the employment of DAGP with an automatic grammar to trim out the size of the domain space. Tavares and Pereira [36] compared the addition of specific type constraints to the classic GP algorithm and, as expected, STGP improved the quality of evolved individuals.

An increase in individuals' size and complexity during the evolutionary process without a simultaneous improvement in fitness is another common phenomenon in GP, known as bloat. This scalability issue results in an increase in both evaluation time and memory usage. Also, it has strong links with the size of the population and its dynamics. Furthermore, since the population has an essential role in guiding the search in the solution space, this issue can be critical to solving.

Several researchers tried to cope with this limitation. Wong and Zhang [37] performed an online simplification of the programs to reduce the size and the redundancies of evolved programs while improving their interpretability. Ferreira, Figueira and Amorim [6] included a bloat control factor in the objective function to penalize the tree size. Jackson [38] explores a strategy to improve diversity by eliminating individuals with duplicate fitness values in the initial population and attempting to maintain it throughout the rest of the evolutionary process. It produced notable improvements over the standard GP. Finally, Vega et al. [39] proposed a new approach to deal with bloat focused on the time required for the evaluation of individuals' fitness instead of being focused on size growth. This technique was employed in a variety of real-world problems showing positive results.

The work presented by Vanneschi, Castelli and Silva [40] is helpful for a better understanding of the concepts of bloat, overfitting and functional complexity in GP solutions and introduces new intuitive measures to quantify them.

Several authors looked for ways to improve the algorithm's efficiency. Typically, one of the most time-consuming components of GP is fitness calculation. To speed up this process, Hildebrandt and Brake [41] presented a way of using surrogate models in GP, resulting in an improvement in convergence speed and solution quality. This approach avoids the computation of non-promising individuals and is particularly interesting when having a simulator with an expensive fitness calculation.

Being conscious of standard GP failures and possible methods of overcoming them are essential aspects to have in mind during the development of this dissertation since they can be related to emerging difficulties.

2.1.3 Genetic Programming algorithms

GP is known for being computationally expensive to solve, requiring large amounts of resources to be executed. Every member of the population needs to be evaluated and tested against the objective, and thus, fitness calculation is the most computationally expensive operation in the procedure. Throughout the years, several technologies have been conceived to address this issue and create efficient GP algorithms that can be used to solve a variety of problems.

ECJ is an evolutionary computation toolkit written in Java created by Luke [42]. It is an open-source framework that supports a variety of techniques for expressing algorithms, including GP. Also, because most of its functionalities are defined by a set of parameter files prepared by the user, it is a highly flexible framework suitable for complex problems.

Baeta et al. [43] performed a comparative performance study between the most commonly used GP algorithms among the scientific community. The obtained results indicate that TensorGP GPU achieves shorter execution times when compared to other engines under the same controlled setup. Also, the GP algorithm of ECJ achieves the best performance in this study regarding traditional GP evaluation. Developed by Baeta et al. [44], TensorGP is a GP algorithm based on data vectorisation, written in Python and uses the TensorFlow library. The fast execution times achieved by this vectorisation algorithm show its promising ability to solve large-domain problems.

In [45], a symbolic regression benchmarking framework is introduced to compare the performance of different methods on a range of regression problems. Considering both the accuracy and simplicity of the models, the results show that GP-GOMEA (Gene-pool Optimal Mixing Evolutionary Algorithm for Genetic Programming) is one of the algorithms with the best performance. This algorithm, introduced by Virgolin et al. [46], is written in C++ under the hood for speed. It is considered proficient in finding models with short symbolic expressions that can frequently be interpreted. This algorithm can be used as the classic tree-based GP or as the innovative GP-GOMEA, bringing the advantages of a more effective exploration of the search space of solutions and an evolutionary process less prone to bloat. The variation operator, GOM, introduced by GOMEA, extensively explores the search space by combining partial solutions iteratively to improve an individual. In contrast to the subtree crossover used in the standard tree-based GP, GOM mixes tree nodes instead of subtrees. Also, it guarantees that an offspring is at least as fit as the parent.

2.2 Vehicle Routing Problem - VRP

The Vehicle Routing Problem statement was first presented in [47] as a generalization of the Traveling Salesman Problem (TSP) [48] with the new concept of a multiple-route node service. VRPs can be described as a problem of finding optimal routes for multiple vehicles that visit a set of customers geographically distributed.

The problem is formally defined as a graph $G = (V, E)$ where $V = \{v_0, ..., v_n\}$ is the set of vertices, with v_0 being the depot vertex and the remaining vertices customers that need to be serviced. The edge set $E = (v_i, v_j)|(v_i, v_j) \in V^2, i \neq j$ states for the paths between customers and $c(v_i, v_j)$ for the travel time, distance or travel cost between v_i and $v_j, \forall v_i, v_j \in V$. Each vehicle has a limited capacity, and each customer is visited only once. The primary objective of this problem is to minimize the cost function of the total distance travelled by all vehicles [49].

Although the minimization of the total travelled distance is the most usually considered optimality criterion in the VRP literature, other objective functions have been proposed. Some examples are the minimization of the total route time and the fleet size or the maximization of the quality of service and the balance between routes. Jozefowiez, Semet and Talbi [50] give an overview of multi-objective VRPs.

Bräysy and Gendreau [51] described the set of criteria that must be considered when evaluating a solution to a VRP problem. A good solution to the problem must englobe a trade-off between execution time, solution quality, ease of implementation, robustness and flexibility. First, the time it takes to produce solutions to the VRP can be crucial, especially when considering real-world cases. Also, the quality of the generated solution, estimated by the objective function, is evidently a key component. As for flexibility, a technique should be easily adapted to changes in the simulator, the problem features or the objective function. Concerning robustness, a solution to the problem must be suitable across all problem instances. Accordingly, considering the GP approach developed in this dissertation, an acceptable trade-off between all these criteria is expected to be achieved.

New challenges and practical requirements have risen in the industry, making it a necessity to extend the classical VRP definition into new formulations of the problem. Pillac et al. [49] listed some of the most studied variations: the Capacitated VRP (CVRP), where vehicles' carrying capacity is limited; the VRP with Pick-up and Delivery (VRPDP), where vehicles need to visit both pick-up and drop-off locations; the Heterogeneous Fleet VRP (HFVRP), where vehicles may have different carrying capacities; the VRP with Time Windows. (VRPTW), where each delivery must be made within a specified time frame. In this project, we will focus on the latter.

2.2.1 Vehicle Routing Problem with Time Windows - VRPTW

The VRPTW aims to find a set of delivery routes so that each node gets visited within its time window while respecting vehicles' capacity. The main goal is to minimise the travelled distance.

The identification of all business constraints and parameters subject to the routing problem is an instrumental step in the design of a good solution and should be done carefully. However, it

highly depends on the considered scenario. In order to clarify this process, some studies that solve the VRPTW will be presented next.

For instance, a recent paper published by Pan, Zhang and Lim [52] investigated a routing problem considering as features a time-dependent travel time, time windows, multiple trips per vehicle, a necessary loading time at the depot and a limited journey duration. Also, the authors emphasized the point of considering time-dependent travel times. Although the classic assumption within common studies is that the time it takes to travel between two customers is constant and independent of the time of departure, this point may not occur in real scenarios since travelling velocity significantly varies throughout the day in urban areas (peak and off-peak hours). Accordingly, since this dissertation aims to incorporate real-world data of a large European online retailer that operates in urban areas, considering this assumption could be beneficial.

Cömert et al. [53] proposed a hierarchical approach known as the cluster-first and route-second method to solve large size real-world VRPTWs considering two objectives simultaneously: the minimization of both waiting time and total travelled distance.

Arnold and Sörensen [54] proposed a data mining based method to develop knowledge about the structural characteristics of a VRP. This procedure aims to distinguish between good and not-so-good solutions to the problem by analysing the solutions' structure. In the end, the aim was to be able to explain which problem features should be included to make a solution to the problem good. The most important features found by the model are the average distance between connected nodes and the depot, the compactness, the width and the span in the radius of routes.

2.2.2 Dynamic Vehicle Routing Problem with Time Windows - DVRPTW

Most of the studies addressing the VRPTW deal with deterministic and static problems, implying that all problem data is known apriori and that minor changes occur during routes' execution. Nonetheless, in most real-world applications, these problems are dynamic, meaning that new requests may arrive in real-time. Also, due to accidents or unexpected situations, dispatchers frequently need to readjust vehicles' routes to enhance service quality and efficiency. Due to the advances in communication and fleet management systems, people are able to obtain and analyse real-time data quickly. As a result, dynamic VRPs have lately gathered more and more attention.

The VRPs have two dimensions, according to Psaraftis [55]: evolution and information quality. The information evolution refers to data availability along the routes' execution, classified as static or dynamic VRP. The information quality is related to the certainty or not of the information available, resulting in a deterministic or stochastic VRP. Consequently, the dynamic VRPs can be divided into two categories: dynamic and deterministic, and dynamic and stochastic. Psaraftis et al. [56] give an overview and a classification of various dynamic VRP variations.

Two strategies for solving the DVRPTW have been distinguished by Hong [57]. The first one, the forward-looking strategy, seeks to predict future stochastic requests according to the probability of their occurrence. Thus, the unarrived requests will be regarded as known requests in the planned solution. Nonetheless, a large amount of past request data is required for this technique to be computed. For instance, Ferrucci et al. [58] employs this strategy, using previous data to

direct vehicles toward locations where future requests are likely to occur. Since the availability of past information is the only condition to use this method, the authors highlight the applicability of the approach to a wide span of real-world transportation problems. The second strategy described by Hong is the short-sighted strategy where unknown requests are not considered for the initial solution, being added once they arrive during the working day. Examples of studies that applied this method can be found in [18, 19, 59].

An important metric to consider when studying dynamic problems is the degree of dynamism. Ichoua et al. [60] consider two dimensions for this metric based on the frequency of changes (i.e. the rate at which new information is received) and the urgency of requests (i.e. the delay between the moment the request is received and its expected service time). Consequently, some metrics have been proposed and improved over the years to measure the problems' dynamicity.

Lund et al. [61] presented the first measurement for the degree of dynamism of a dynamic problem. Equation 2.1 shows this metric, defined as the fraction of dynamic requests n_d in the total number of requests n_tot.

$$\delta = \frac{n_d}{n_{tot}} \tag{2.1}$$

In order to include the arrival time of requests, a new metric called effective degree of dynamic δ^e was lately proposed. After that, Larsen [62] suggested an effective degree of dynamism for problems with time windows, reflecting the urgency level of customers' orders. This measure is calculated by Equation 2.2.

$$\delta^e_{TW} = \frac{1}{n_{tot}} \sum_{i \in R} (1 - \frac{r_i}{T}) \tag{2.2}$$

Let n_{tot} be the total number of requests, T the length of the planning horizon, and R the group of requests. The reaction time r_i is defined as the interval between the arrival time t_i and the end of the time slot l_i, given a request $i \in R$. Longer reaction times imply greater flexibility in incorporating new requests into current routes. This measure only accepts values in the interval [0, 1] and it increases with the dynamicity level of a problem [49].

Another common topic addressed in dynamic problems studies is the anticipation of future requests by incorporating waiting strategies in the solution. Usually, two strategies are considered: Drive-first, in which the vehicle first drives to the next destination and then waits there, if necessary, and Wait-first, in which, in case the vehicle is expected to arrive before the start of the time window to the next destination, it first waits at the current location. Ichoual, Gendreau and Potvin [63] affirm that if the problem has a dynamic setting, it is better to adopt a wait-first strategy since, if new requests occur before departure, there is still a chance to rearrange the next destination. Also, the obtained results in [23] showed that the wait-first strategy is usually more compensatory than the drive-first strategy.

By involving dynamic information in these problems, the complexity of their decisions is increased, and new challenges appear [64]. One of these challenges is the development of effective

solution techniques which incorporate real-time information while meeting the time restrictions imposed by continuously evolving environments [63].

Armas and Melián-Batista [59] solve two variants of real-world DVRPTW using a continuous re-optimisation method in which the optimisation is carried out each time a new customer request arrives by updating the current solution. In contrast, Chen et al. [18] utilise a periodic optimisation strategy to solve a dynamic VRP that consists in partitioning the whole time horizon into equal time slices, followed by periodically solving a sequence of static VRPs at the beginning of each time slice.

Jacobsen-Grocott et al. [23] tried to solve a DVRPTW based on a decision-making process in which each time a new order is received, a tentative to re-generate the planning routes is made based on the decision to either accept or reject the order. A meta-algorithm is utilised to maintain a set of routes and update it by heuristic in case of a newly admitted request.

There has been plenty of research for solving the VRPTW, and many heuristics have been proposed. However, although those algorithms can generally find optimal solutions for the problem, they are unable to meet the conditions of the DVRPTW since they require long computational times to be solved.

2.2.3 Conventional solution approaches

The incorporation of real-world data into VRPs involves complex constraints that introduce significant complexity, making it computationally expensive to solve. Therefore, solving VRPs requires the use of advanced algorithms. The most common strategies to solve VRPs are exact methods, heuristics and meta-heuristics.

Exact methods (e.g. mathematical programming, constraint programming and dynamic programming) are often used to solve VRPs. However, although they provide optimal solutions, they can only handle small-size instances due to the high complexity of the problem. On the other hand, heuristics and meta-heuristics can efficiently support large-scale problems but typically yield near-optimal solutions, being considered approximate methods [22].

Heuristic strategies are focused on systematically finding an acceptable solution for a specific problem in a limited time and are usually faster than exact methods. Heuristic techniques can be categorised into construction heuristics and improvement heuristics. Construction heuristics are commonly employed to design initial solutions to the problem, which are later enhanced by applying improvement heuristics or meta-heuristics. Improvement heuristics are typically applied to solutions that exact methods or other heuristics have already developed. Commonly used route construction heuristics in the operations research community are typically based on the well-known route-first cluster-second strategy proposed by Solomon [14], the Clarke and Wright [15] savings heuristic, the nearest-neighbour heuristic or the cheapest insertion heuristic both introduced by Solomon [16]. More details regarding construction and improvement heuristics can be found in [51]. As a generalisation of classical heuristics, meta-heuristics are procedures that provide a deep exploration of the solution by recombining solutions and rules. Typically, they produce much higher quality solutions than classical heuristics yet, within an increased computing time.

Unfortunately, meta-heuristics are context-dependent techniques, making them difficult to extend to different problem domains. Many meta-heuristics have been successfully applied for the VRPs, including local search methods (e.g. Tabu Search [17] and Simulated Annealing [65]), population search (e.g. genetic algorithms [66]) and learning mechanisms (e.g. ant colony optimization [19]).

Hyper-heuristics, such as GP, are another method that has been growing research in solving optimisation problems, which go a step beyond meta-heuristics. While meta-heuristics search within a search space of problem solutions, hyper-heuristics' search space is composed of heuristics or meta-heuristics, meaning that heuristics are used to search for heuristics. This method can usually provide much better solutions than heuristics or meta-heuristics, making it a revolutionary technique that can be very useful for solving dynamic VRPs.

For instance, an example of a Large Scale VRP is presented by Xiao et al. [67] in which is suggested an evolutionary multi-objective route grouping method to partition the large dataset into smaller subcomponents. The decomposition is performed by a divide-and-conquer framework that simultaneously considers three objectives: inter-group distance, intra-group distance, and balance in size. After finding the clusters, a Tabu Search algorithm is used to optimise them. Generally, the developed approach shows competitive results. However, since this method considers a heuristic algorithm, domain knowledge is required to find good heuristics. Therefore, the use of GP to automatic evolve the heuristics would be advantageous.

Reinforcement learning (RL) has also been considered a promising tool to solve VRPs. Hildebrandt, Thomas and Ulmer [68] summarise existing solution approaches for solving stochastic dynamic VRPs in both operations research and computer science scientific communities and highlight the potential of combining them with RL to improve the solution's quality. The authors propose a high-level concept on how to incorporate them to solve stochastic dynamic VRPs by searching the action space and evaluating actions based on state information. Although RL-based solutions' potential is evident in solving these transportation problems, little research has been done on the topic.

For instance, Nazari et al. [69] introduce a procedure based on RL that trains a neural network policy to construct the route from scratch. This procedure appears to be effective for both static and dynamic VRPs. It uses a verifier to locate feasible solutions and a reward signal to indicate how well the route construction is going. Compared to classical heuristics, this approach has the advantage of being effective for large problem sizes and generally achieving superior performance. However, when considering the goal of this dissertation of producing an explainable approach to tackle the VRP, this procedure is not compatible due to its black-box nature.

Since VRPs are NP-hard, one of the main challenges in VRPs is finding the balance between available CPU time, problem size and solution quality. The constant development of the techniques described above helps in achieving better results while reducing computation time [22].

2.3 Genetic Programming for combinatorial problems

The ability of GP to introduce explicit models for linear and nonlinear processes in a wide range of problems has triggered research in the area. Despite increasing GP interest application in many combinatorial problems, almost no research has been done on applying it to VRPs. This section aims to describe the successful application of GP in literature studies, encouraging its application to VRPs. Additionally, the GP parameters used in these studies will be analysed and evaluated for their adaptability to our problem.

The Job Shop Scheduling problem is an example of a problem that has been gathering increasing attention to be solved using GP approaches. For instance, in [6], the authors solve a Dynamic Job Shop Scheduling problem with a GP iterative approach, where reasoning on the problem guides the evolutionary algorithm to evolve better dispatching rules. Also, features from existing rules in the literature are included in algorithm parameters. The results showed that the best-evolved rules' performance increased by 19% compared to the best possible combination of benchmark rules. Note that some GP terminals used in the scheduling literature can be adapted for the DVRPTW. These include the critical ratio, calculated by the ratio between the slack time and the time needed to serve the client, and the utilisation feature, which in our problem could symbolise the number of customers allocated to a route.

In the family of knapsack problems [70], GP has also been considered a technique capable of producing competitive solutions. Drake et al. [26] used a GP based hyper-heuristic to produce constructive heuristics for the multidimensional 0-1 knapsack problem.

Another problem in which solutions have been gaining a considerable boost in performance and interpretability with GP is the arc routing problem [71]. This problem is equivalent to the VRP but serves edges instead of nodes. In [25], Wang et al. use GP to solve the Uncertain Capacitated Arc Routing Problem (UCARP), with the improvement of including a niching technique, i.e., a technique which groups the individuals with the same phenotypic behaviour into niches and selects the smallest, to simplify the routing policies in the evolutionary process. The results show a significant improvement in test performance and interpretability compared to other GP approaches for UCARP. A commonly considered GP terminal in arc routing problems, which could be adapted to our problem, is the extra cost of serving the candidate client.

The success of GP approaches in combinatorial problems is notorious. Moreover, the most effective applications are generally for Dynamic or Uncertain problems, which are the settings in which the classical optimisation methods based on mathematical programming or meta-heuristics fail more frequently. Since GP can provide decision rules in real-time as new problem information is revealed, its competitiveness is increased against other approaches.

2.4 Genetic Programming for Vehicle Routing Problems

After presenting the GP approaches and its successful application in many combinatorial problems, this last section exposes the literature that applies them to the VRP.

Since the VRP is an NP-hard problem, exact solvers may fail to provide good solutions in a reasonable time. This is even more evident for the dynamic VRP, as decisions need to be made in real-time. Usually, heuristic methods (classical heuristics and meta-heuristics) are used to find high-quality feasible solutions in a limited time that can be applied to real-world problem instances. However, these methods have their limitations when applied to different problem data sets since they need to be manually adapted to each class of problem. Hyper-heuristics, such as GP, can be used to automate the heuristic design process, providing a broader solution for any instance domain. However, applications of GP to the dynamic VRP are scarce in the literature.

The work presented by Costa, Mei and Zhang [24] is an interesting example to explore since it considers a Large Scale VRP which is commonly the case of real-world scenarios. A GP approach is proposed, and the authors try to identify more promising areas of the search space by employing a method similar to the one explained previously from [54], i.e. a Knowledge Guided Local Search method. Obtained results show the effectiveness of this procedure. However, the approach conducted only considers solving a static VRPTW.

Jacobsen-Grocott et al. [23] consider solving the dynamic VRPTW using a GP Hyper-Heuristic (GPHH). This paper is probably the most similar to the work of this dissertation. A meta-algorithm is presented to maintain a group of routes along the scheduling horizon. When a new request arrives, the meta-algorithm attempts to re-generate them based on current vehicle states and previous requests. Afterwards, some heuristics are manually designed whose performance is compared to the proposed GP algorithm that automatically evolves heuristics. The results show that the heuristics evolved by GP performed significantly better than manually designed heuristics. Also, such an advantage becomes more notorious as the degree of dynamism of the VRP increases. Although the instances used in this paper are not part of real-world data, the different classes of dynamism tested, as well as the positive results obtained, are encouraging.

Despite the scarcity of studies in literature where GP-based approaches are applied to VRP instances, the works presented in [23] and [24] provide promising results. For these reasons, this dissertation aims to explore the main advantages of GP when solving VRPs, contributing to the lack of search made on the topic.

Chapter 3

Problem Description and Solution Approach

The VRP has played a vital role in the transportation sector, and many researchers have made remarkable achievements in solving it. However, most of them focused on the static version of the problem, although most VRPs are dynamic in the real world (e.g dial-a-ride services [72], meal or instance delivery [13] or emergency services [73]). The dynamic VRP adds to the static VRP the importance of the time dimension, the uncertainty of future events, the need for rerouting, and faster algorithm runtimes. The latter topic fails to be achieved by traditional methods (exact methods, heuristics and meta-heuristics) since they require long computational times to be executed. GP is an approach with large potential to solving dynamic VRPs due to its ability to evolve expressions that can solve VRPs in fractions of a second. In this dissertation, we focus on the particular case of DVRPTW, i.e., dynamic VRPs considering delivery time windows.

In this chapter, we formalise the problem by introducing a mathematical formulation that models the static version of the VRPTW (Section 3.1). Afterwards, we present the proposed solution approach, which learns routing policies described by symbolic expressions (Section 3.2). This approach is based on a GP algorithm that evolves symbolic expressions while evaluating their quality (fitness) through a simulation model.

3.1 Problem Statement

Formally, a DVRPTW can be stated as a connected graph $G = (V, E)$ where $V = \{v_0, ..., v_n\}$ is the set of vertices, with v_0 being the depot and the remaining vertices customers that need to be serviced. The edge set $E = (v_i, v_j)|(v_i, v_j) \in V^2, i \neq j$ states for the paths between customers and t_{ij} for the travel time of the shortest path between v_i and v_j, $\forall v_i, v_j \in V$. Also, the distance between two vertices is represented by d_{ij}.

The depot has an associated time window [0, W] that corresponds to the length of a working day. All vehicles must leave and return to the depot within the time horizon. The customer requests known at the beginning of the planning horizon are called static customers, represented as V_s. The

remaining customer requests, denoted as V_d, constitute the set of dynamic customers who are revealed during the planning horizon. Accordingly, the set $V' = V_s \bigcup V_d = \{v_1, ..., v_n\}$ represents all customer requests of a day of work. A solution to the problem is a path in G, starting from v_0, sequentially visiting specific customers and returning to v_0.

Each customer is characterised as $v_i = (x_i, y_i, l_i, u_i, q_i, s_i, a_i)$, with $l_i \leq u_i$ and $0 \leq a_i \leq W$. In this vector (x_i, y_i) represents the location of the customer, l_i and u_i the lower and upper limits of the time window, q_i the demand of items to be delivered, s_i the duration of the service and a_i the time when the customer request became available. Since a wait-first strategy is adopted, a vehicle k must wait if it arrives at client v_i before l_i. It is assumed that a vehicle cannot serve a customer v_i before l_i but that it can serve it after u_i, i.e. delayed, but it also means a penalization. One should note that delays are only allowed at the customers, as all the vehicles must return to the depot before W. There are K vehicles with a limited capacity Q and each customer is visited only once by one vehicle in a working day.

There are two main binary decision variables to consider in the model: X_{ijk}, which defines each arc that is travelled by the vehicle k; Y_i, which represents whether customer i is served within the time horizon, i.e. if the customer is accepted. Additionally, two continuous decision variables are used: ϕ_i^+ represents a late arrival at node i; b_i indicates the time at which the service began at node i. The proposed formulation reads as follows:

Sets

V	Set of vertices
V'	Set of vertices to be serviced
E	Set of edges
K	Set of vehicles

Cost Parameters

α	Average cost per distance unit
β	Average cost of rejecting a customer

System Parameters

d_{ij}	Travelled distance between vertices i and j
t_{ij}	Travelled time between vertices i and j
l_i	Lower limit of vertex i's time window
u_i	Upper limit of vertex i's time window
q_i	Demand of vertex i
a_i	Available time of vertex i
s_i	Service time of vertex i
Q	Vehicles limited capacity
W	Length of a working day

Decision variables

$X_{ijk} = \begin{cases} 1, & \text{if arc } (i,j) \text{ is travelled by vehicle } k \\ 0, & \text{otherwise} \end{cases}$

$Y_i = \begin{cases} 1, & \text{if client } i \text{ is accepted to be served in that working day} \\ 0, & \text{otherwise} \end{cases}$

ϕ_i^+ Lateness at node i

b_i Time at which node i starts being served

Mathematical Formulation

VRPTW:

$$min \quad \alpha \sum_{(i,j)\in E} \sum_{k\in K} d_{ij} X_{ijk} + \beta \sum_{i\in V'} (1 - Y_i) \tag{3.1}$$

$$\sum_{i\in V} X_{ijk} = \sum_{i\in V} X_{jik} \quad \forall j \in V', k \in K \tag{3.2}$$

$$\sum_{k\in K} \sum_{j\in V} X_{ijk} = 1 \quad \forall i \in V' \tag{3.3}$$

$$\sum_{j\in V'} X_{0jk} = \sum_{i\in V'} X_{i0k} \quad \forall k \in K \tag{3.4}$$

$$\sum_{j\in V'} X_{0jk} \leq 1 \quad \forall k \in K \tag{3.5}$$

$$\sum_{i\in V'} \sum_{j\in V} q_i X_{ijk} \leq Q \quad \forall k \in K \tag{3.6}$$

$$b_i + s_i + t_{ij} \leq b_j + M(1 - X_{ijk}) \quad \forall i, j \in V', k \in K \tag{3.7}$$

$$l_i \leq b_i \quad \forall i \in V' \tag{3.8}$$

$$b_i \leq u_i + \phi_i^+ \quad \forall i \in V' \tag{3.9}$$

$$X_{ijk}, Y_i \in \{0,1\} \quad \forall i, j \in V, k \in K \tag{3.10}$$

$$\phi_i^+, b_i \geq 0 \quad \forall i \in V \tag{3.11}$$

The objective function 3.1 aims to minimise the total cost of the routes. Namely, the total travelled cost and the cost of rejected customers, i.e. the customer requests postponed to the next working day. Constraints 3.2 are flow conservation constraints, meaning that the number of departures in each vertex must be equal to the number of arrivals. Constraints 3.3 guarantee that

each client is visited only once by one vehicle. Constraints 3.4 establish that each route starts and ends at the depot. Constraints 3.5 exist to ensure that one route departs at maximum once from the depot during the working day. Constraints 3.6 are capacity constraints, assuring that the maximum capacity of the vehicles is not violated. Constraints 3.7 are subtour elimination constraints [74], assuring that the service start time at a node is greater than or equal to the service start time at the previous node plus its service time and the travel time between the two. M is a constant of very high value. Constraints 3.8 and 3.9 impose the time window constraints. Finally, Constraints 3.10 and Constraints 3.11 define the domain of each decision variable.

3.2 Solution Approach

This section describes the proposed methodology for developing the GP algorithm capable of generating solutions to the problem. The basic concept of the proposed approach is to evolve routing policies that are able to build the solutions for the DVRPTW. Inspired by the work presented in [6], a guided empirical learning method is proposed, where human intelligence is involved in the problem's reasoning to guide an iterative algorithmic search. At each iteration, the terminal or function set suffers modifications, and new solutions are generated. Figure 3.1 presents the procedure adopted for the development of the solutions.

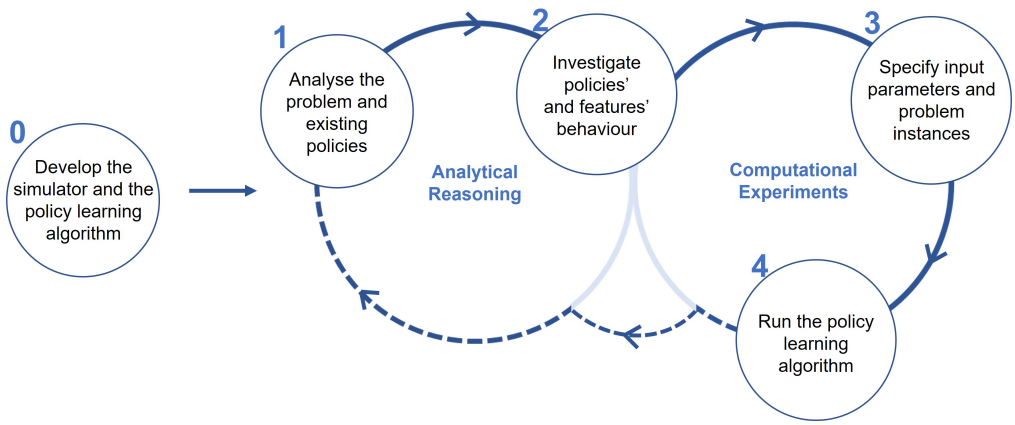


Figure 3.1: Solution approach methodology.

The developed approach is divided into two main parts: the analytical reasoning on the problem, providing all necessary inputs to initialise the GP algorithm (problem parameters and business constraints, covered in Section 3.1) and the computational experiments part, englobing both the simulator and the policy learning method. First, the policy learning algorithm and the DVRPTW simulator are employed (step 0). Next, an iterative process starts. In step 1, the problem and existing policies are investigated, and all the relevant parameters and constraints are extracted. In step 2, features' behaviour is analysed and conclusions about their importance in the evolutionary process are drawn. Then follows step 3, where problem parameters, GP terminals or operators

may suffer modifications. Finally, in step 4, the GP algorithm runs, new routing policies are generated, and the process may follow into step 1, beginning a new iteration. When the best learning configuration is achieved, the last GP run is performed, and the final policies are extracted.

Individuals in GP are typically represented as symbolic expression trees and correspond to the composition of terminals (leaf nodes) and functions (internal nodes). Figure 3.2 depicts an example of an individual and its corresponding routing policy. In this example, *RTT*, *NRTT*, *CR* and *C* are terminals and *, / and - functions, i.e. mathematical operators.

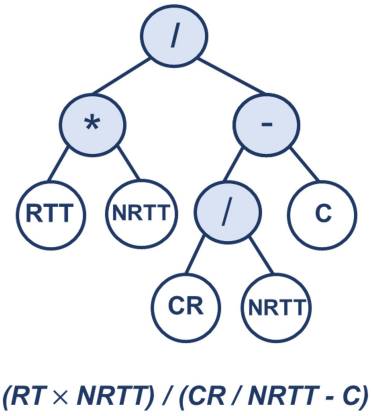


Figure 3.2: A GP individual and its corresponding routing policy.

Next, in Section 3.2.1, the DVRPTW simulation model is detailed. After that, Section 3.2.2 describes the proposed GP algorithm and explains the calculation of the fitness metric.

3.2.1 DVRPTW Simulation Model

A simulation model was designed to evaluate the performance of a GP individual on the considered problem (Figure 3.3). The DVRPTW is solved as a sequence of static VRPTWs, the so-called periodic optimisation strategy [18]. Accordingly, a working day is divided into n_{t_s} equal-length time slices, and, in each one, a static version of the problem is solved for the group of currently known customers, following the works [18, 75].

Algorithm 1 presents the outline of the DVRPTW simulator. Initially, a solution is generated for the static customers. After that, whenever a new time slice starts (Algorithm 1, line 11), a new solution is developed if further requests have arrived. In this case, with the function *getAvailableCustomers(·)*, we check which customers from the current best solution have not yet been served and can be added to the set of available customers V_{av} and be considered for the rearrangement of the routes to include the newly arrived customers. This procedure is repeated until the end of the working day is reached. A newly generated solution is considered the best so far developed result if the number of rejected requests is less or equal to the number of rejected requests of the current best solution (Algorithm 1, line 15). A customer request can be rejected, i.e.

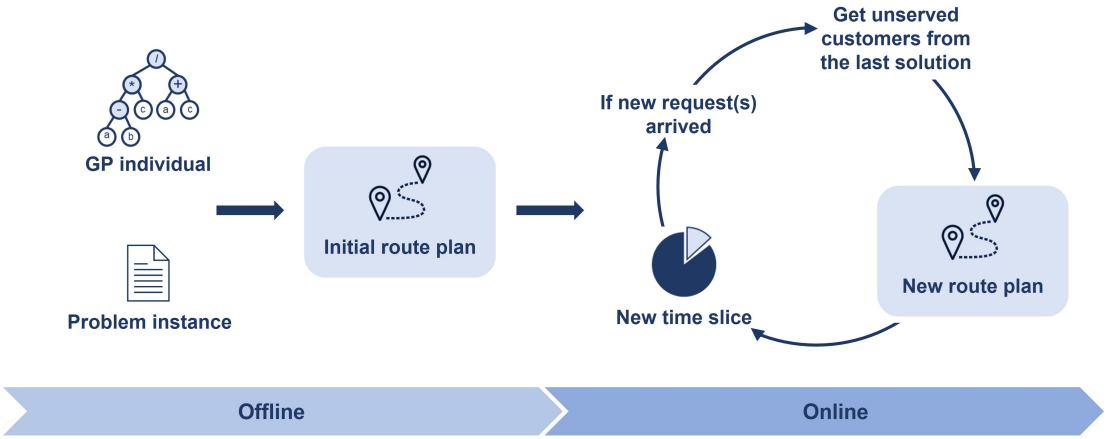


Figure 3.3: Simulation model overview.

postponed to the next working day, in case neither of the vehicles has enough remaining capacity to serve it or if, by accepting it, the worker would arrive at the depot after the end of the working day.

Algorithm 1 : Simulation Model Algorithm, $r \leftarrow runSimulator(i,n)$

Input: Instance i, individual n
Output: Result r

$t \leftarrow 0$
for each $k \in K$ **do**
  $loc_k \leftarrow v_0$
end for
// Generate initial solution
if $V_s \neq \emptyset$ **then**
  $r \leftarrow generateSolution(i,n,t,V_s,K)$
end if
// Decision making process
$t_s \leftarrow 1$
Start timer t
while $t \leq W$ **do**
  if $t \geq t_s \frac{W}{n_{t_s}}$ **then**
    $V_{av} \leftarrow getAvailableCustomers(i,r,t)$
    if *newDynamicClient* **then**
      $r' \leftarrow generateSolution(i,n,t,V_{av},K)$
      if r' better than r **then**
        $r \leftarrow r'$
      end if
    end if
    $t_s \leftarrow t_s + 1$
  end if
end while

A fundamental component of Algorithm 1 is the function *generateSolution(·)* (Algorithm 2). This function encompasses a route construction algorithm responsible for developing a solution to the problem. The procedure builds the routes simultaneously, using the policy evolved by GP. First, considering the GP individual and having the list of available customers and routes, a score is calculated for each pair <client, route> (Algorithm 2, line 9). The lowest value of this set of scores is saved during scores' calculation (Algorithm 2, lines 11 - 13) and, in the end, the corresponding customer is added to the paired route (Algorithm 2, line 23). This procedure is repeated until all customers are added to the routes or rejected due to unfeasibilities. The function *isFeasible(·)* verifies if a customer v is feasible to serve with a route k, i.e. if by serving it, no capacity or depot time constraints are violated. If at the end of the scores' calculation process, the client v' has no client assigned, i.e. it is *null*, it means that there are no feasible clients, and all the available clients should be added to the set of postponed clients V_P (Algorithm 2, lines 18 - 21). During the routes construction process, all relevant information about the result is updated (planned routes, total travelled time and distance, number of delays and number of postponed customers) and, in the end, returned to the leading implementation of the simulator.

The $calculateScore(\cdot)$ function is the most time-consuming part for the simulator since it is the function that is executed most often. So, to try to save some computations of this function, especially at the beginning of the route construction process, before identifying the next client to be accepted, the $getRoutesToUse(\cdot)$ function returns all routes that have clients allocated to them and an extra empty route, if any. In this way, for calculating the scores between pairs, we are only considering the maximum number of needed vehicles and not always the all K vehicles set.

3.2.2 Policy Learning

GP iteratively transforms and improves a population of mathematical expressions in order to solve a problem. The algorithm's search space is the space of all possible mathematical expressions comprising functions and terminals, appropriate to the problem domain. The terminals set reflects the features of the problem state, while the functions set is composed of operators that combine the features in a GP individual. The goal is to find the fittest individual in the search space, where the fitness of each individual is proportional to its ability to solve the problem. The standard GP algorithm starts by generating a random initial population that is evolved over many generations. At each generation, all individuals are evaluated based on their fitness function. In this algorithm, a standardised fitness measure is considered. The best ones are selected, reproduced and combined for the next generations. This process is repeated until the algorithm reaches a termination criterion.

In this work, GP will be used to evolve a population of routing policies executed to solve a DVRPTW. A routing policy is evaluated by applying the corresponding expression to a set of instances, called the training set. For each instance, the simulator is conducted, following the description given in Section 3.2.1. As explained previously, the routing policy is used in each decision point to calculate a score for each pair <client, route>. Next, the minimum of these values is selected, and the customer is joined to the corresponding route.

Algorithm 2 : $r' \leftarrow generateSolution(i,n,t,V_{av},K)$

Input: Instance i, individual n, time t, set of available customers V_{av} and set of vehicles K
Output: Result r

```
1:  minScore ← ∅
2:  routesToUse ← ∅
3:  Nclients ← size(V_av)
4:  for j ← 0 to Nclients do
5:      routesToUse ← getRoutesToUse(K)
6:      for each k ∈ routesToUse do
7:          for each v ∈ V_av do
8:              if isFeasible(k,v) then
9:                  score' ← calculateScore(k,v,i,n)
10:                 if score' < minScore then
11:                     minScore ← score'
12:                     k' ← k
13:                     v' ← v
14:                 end if
15:             end if
16:         end for
17:     end for
18:     if v' is null then
19:         V_P ← V_av
20:         break
21:     end if
22:     remove customer v' from V_av
23:     add customer v' to route k'
24: end for
```

Algorithm 3 summarises the steps of the proposed GP. In each generation g a whole population P_g is evaluated and updated. Each individual n in the population represents a routing policy. Firstly, the initial population is generated using the method Ramped Half-and-Half [10], which combines the grow and the full methods with equal probabilities (Algorithm 3, line 1). The full method creates trees where the specified maximum depth equals the path length between the root and every terminal. Instead, the grow method creates trees of varying sizes and shapes.

For each instance of the training set T, the fitness is computed by running the DVRPTW simulator. The simulator returns the resulted fitness when applying policy n for instance i. This value is calculated by adding the costs of the total travelled distance and the number of rejected customers. Additionally, a delay avoidance strategy is also considered, adding a penalty component to the equation. Equation 3.12 illustrates the calculation of this value, where α represents the cost per distance unit travelled, β the cost of rejecting a customer and M a constant of very high value.

$$fitness(i,n) = \alpha n_{distance} + \beta n_{rejected} + M n_{delays} \tag{3.12}$$

The fitness function for the individual n encompasses the average fitness of the individual across all training instances. Equation 3.13 presents the calculation.

$$fitness(n) = \frac{\sum_{i \in T} fitness(i,n)}{|T|} \tag{3.13}$$

Since the objective function of the problem aims to minimise the overall routes' cost, the fitness function follows the same paradigm, meaning that smaller fitness values correspond to better routing policies. At the end of each generation, the best individual in the population is saved.

In Algorithm 3, line 12, the population of the next generation is updated. First, the elitist individuals pass directly to the next generation, based on the elitism parameter. Next, a Tournament Selection procedure starts by randomly selecting a set of individuals from the population based on their fitness. From these, a small percentage is copied to the next generation without suffering genetic modifications (reproduction). The others undergo genetic changes. The crossover operator selects two individuals to be parents and produces two offspring policies. In contrast, the mutation operator creates a new individual by replacing a sub-tree with a randomly generated one.

This process is repeated until a termination criterion is fulfilled, i.e. the maximum number of generations (*maxG*) is reached. The best individual who appeared in any generation is assigned as the result of the GP algorithm. This result is a solution to the problem. After finishing the algorithm's evolution part, the best expression found will be tested in a set of unseen instances called the validation set.

Algorithm 3 : Genetic Programming algorithm

Input: Training set T
Output: Individual n'

1: $P_g \leftarrow InitializePopulation()$
2: $g \leftarrow 1$
3: **for** $g = 1$ **to** $maxG$ **do**
4: | **for each** *individual* $n \in P_g$ **do**
5: | | **for each** *instance* $i \in T$ **do**
6: | | | $fitness(i,n) \leftarrow runSimulator(i,n)$
7: | | **end for**
8: | | $fitness = \sum_{i \in T} fitness(i,n)$
9: | | $fitness(n) \leftarrow standardizeFitness(fitness)$
10: | **end for**
11: | $n' \leftarrow argmin_{n \in P_g} fitness(n)$
12: | $P_{g+1} \leftarrow genetic\ operators\ in\ P_g$
13: **end for**

Chapter 4

Computational Experiments

This chapter starts with a comparison of different GP algorithms, in order to assess which one is more suitable for this problem. The algorithms are evaluated in a simulator for the TSP with time windows (TSPTW). Then, the best algorithm is selected and used in all the remaining experiments (Section 4.1). After that, the DVRPTW instances used are detailed (Section 4.2), followed by the exploration of the tests executed to design the experiments and find the best input parameters configuration (Section 4.3). Finally, the best-obtained routing policies are enumerated and evaluated in unseen instances from the literature (Section 4.4).

4.1 Algorithm Selection

This section will start by giving an overview of the different GP algorithms considered for the comparison (Section 4.1.1). After that, the developed TSPTW simulator and the generated problem instances are detailed (Section 4.1.2). Finally, the comparison is performed, and the GP algorithm is selected (Section 4.1.3).

4.1.1 Considered GP algorithms

The first step in the implementation phase was to enquire which GP Algorithm, among the ones available in the literature, is the most adequate to find a solution for the present problem. This comparison was made based on two criteria: performance and validation time. Apart from that, an essential criterion in selecting the algorithm lies in the fact that they are free and open-source. With the review done in Section 2.1.3, three GP algorithms were considered for the comparison, ECJ-GP [42], TensorGP [44] and GP-GOMEA [46]. Table 4.1 shows the considered GP algorithms with the corresponding implementation language and device support.

ECJ [42] has the advantage of being a well-established framework in the Evolutionary Computation community, with an available manual with all documentation about its functionalities. Also, its high flexibility is promising for computing complex problems. This framework offers a variety of algorithms, including GP, which will be called ECJ-GP.

Table 4.1: Considered GP algorithms along with their respective programming languages and device support.

Algorithm	Language	Processor
ECJ	Java	CPU
TensorGP	Python	CPU/GPU
GP-GOMEA	C++	CPU

TensorGP [44] uses data vectorization to accelerate the evaluation speed. Besides that, it includes the additional feature of permitting the execution in GPU processors, which may promise even lower execution times.

GP-GOMEA [46] is the result of combining the classic tree-based GP with the GOMEA that aims to improve the evolution process performance and exploit the structure of the problem. In addition to that, it is a strategy less prone to bloat.

4.1.2 TSPTW Simulation Model

To evaluate the performance of the GP algorithms, an adaptation of the simulator (described in Section 3.2.1) for the TSPTW was developed in each one of the programming languages. The aim was to test each policy learning algorithm with a simplified version of the actual problem of this dissertation, the DVRPTW. The implemented TSPTW simulator follows the same logic as the simulation model described in Section 3.2.1. Still, K includes only one vehicle and the solution is generated only once at the beginning of the planning horizon.

Instances of the TSPTW were generated to be used in the train and validation phases. The training instances are used in the algorithm's evolution process, while the validation instances are used to evaluate the best-evolved routing policies in unseen instances of the problem. Each instance includes the time window of each client and the matrices of distances and travel times between each client and the depot vertex. Ten different instances were created for each number of customers (10, 20, 30,..., 100). The features of each instance were randomly generated and are proportional to their *id*, where the *id* of an instance corresponds to the order in which they were generated. The distance between two vertices was taken as the travel time between them multiplied by 100. The average travel time is 10, and the average time window size is 100. This process resulted in 100 TSPTW instances, split 20% into train and 80% into the validation set. Table 4.2 summarises the parameters of the generated TSPTW instances.

4.1.3 Algorithms Comparison

In order to achieve comparable results, the GP parameters were set equal for the different engines. Nevertheless, since GP involves stochastic processes in the evolution process (e.g. generation of the initial population, selection and genetic operators), it is difficult to ensure the exact replication of an evolutionary path in different engines even when using the same random seed. However, various experimental runs can be performed to standardise the set of parameters to average out the

Table 4.2: TSPTW instance parameters.

Parameter	**Description**	**Values**
|V|	Number of customers	10,20,30...100
l_i $\forall i \in V$	Lower limit of a node time window	[0, id]
u_i $\forall i \in V$	Upper limit of a node time window	l_i + [0, id*100]
d_{ij} $\forall i, j \in V$	Travel distance between two nodes	[0, id*10]
t_{ij} $\forall i, j \in V$	Travel time between two nodes	d_{ij}*100
|T|	Number of training instances	20
|V|	Number of validation instances	80

inherent variability. In these tests, five independent runs were performed for each GP algorithm. Table 4.3 shows the experimental parameters used.

Table 4.3: Experimental GP parameters for the algorithms' comparison experiment.

Parameter	**Value**
Maximum generations	50
Population size	500
Tournament size	7
Elitism	10
Crossover rate	0.9
Mutation rate	0.1
Initial generation method	RHH
Initial maximum tree height	5
Maximum tree height	17

The set of functions and terminals used in the evolutionary process to construct the routing policies is shown in Table 4.4. These features were selected based on a direct analysis of the problem and extracting the standard parameters that condition it. The aim was to maintain a simple set of features of the problem since, in this analysis, we are focused on testing the algorithms in the same circumstances and not on finding optimal solutions. The slack time corresponds to the difference between the end of the time window and the current time in the routes' construction process. The fitness function is the same presented in Equation 3.13 but, in this case, it does not include the component for the cost of rejected customers since this is not a feature considered in TSPTW. For the sake of simplicity, the average cost per distance unit, α, is set to one for these experiments.

The computational tests were performed on a AMD Ryzen Threadripper PRO processing unit running at 4.2GHz, within a Ubuntu 20.04 virtual environment with 8 CPUs and 44GB, executing with a single thread. For the execution phase, we consider the following five algorithms: ECJ-GP, GP-GOMEA using simply tree-based GP (i.e. with GOMEA turned off), GP-GOMEA using GOMEA, TensorGP executing on CPU and TensorGP running on GPU. Figure 4.1 presents the evolution of the fitness of the best individual across generations for each of the executed GP

Table 4.4: List of GP functions and terminals used for the algorithms' comparison experiment.

Parameter	**Description**
Functions	
+	Addition
-	Subtraction
*	Multiplication
p/	Protected Division
Terminals	
STW	Start of the time window
ETW	End of the time window
D	Travel distance
T	Travel time
S	Slack time

algorithms.

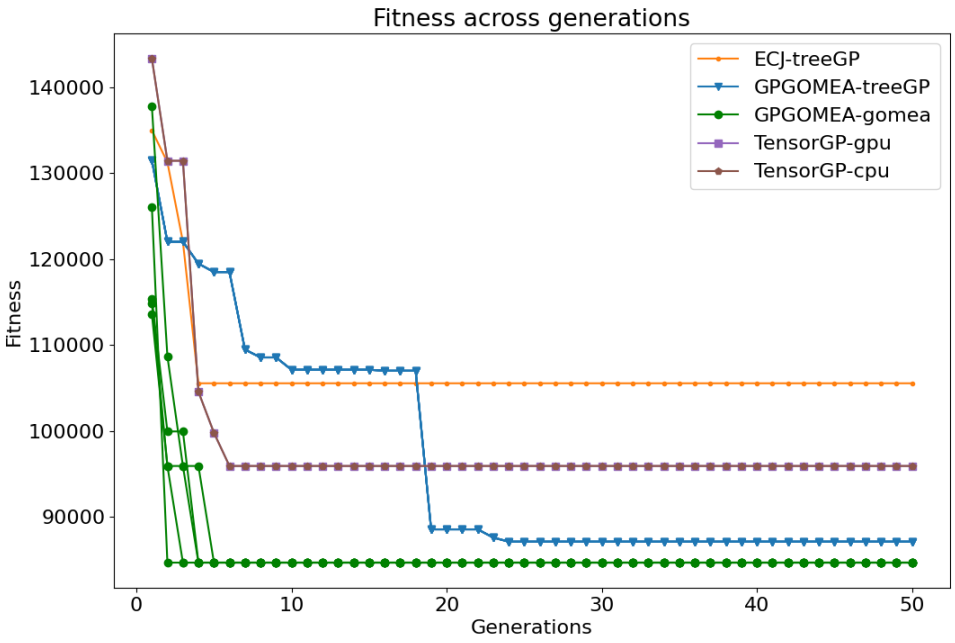


Figure 4.1: Comparing GP algorithms performance across generations.

From the analysis of Figure 4.1, it is possible to conclude that GP-GOMEA, in its two configurations, is the engine that achieves the best performance across the evolution process. Also, comparing its two versions, it is possible to infer that adding GOMEA is advantageous for the algorithm's performance. On the other hand, ECJ-GP includes the algorithm with the poorest performance overall. It is also possible to verify that, for each algorithm, the five independent runs

executed show similar (or equal) evolution progress, achieving the same best individual at the end. Table 4.5 shows the best individual found by each GP algorithm, the number of evaluations performed, and the time required for both the training and validation phases. Cells in bold represent the best individual overall and corresponding fitness, as well as the lowest training and validation times.

Table 4.5: Best individual achieved for each GP algorithm, number of evaluations taken and the training and validation time.

GP Algorithm	Best Individual	Fitness	Evaluations	Training Time (s)	Validation Time (ms)
ECJ	$D+STW*(D+ETW)$	105535	25 000	**84**	439
GPGOMEA-Tree-based GP	$STW*S+D*T-S$	87110	25 499	105	161
GPGOMEA-GOMEA	$STW*S+2*D$	**84675**	174 362	2393	77
TensorGP - GPU	$STW+T*T$	95915	25 000	2530	**56**
TensorGP- CPU	$STW+T*T$	95915	25 000	2571	58

The best routing policy was achieved by GP-GOMEA with GOMEA turned on. This result is in line with what was expected. The high number of evaluations that this algorithm performs (almost seven times more when compared to the other engines) due to the variation operator (GOM) should logically result in a better solution to the problem. Evidently, running this GP algorithm with GOMEA turned on requires heavier computational resources.

Excluding GOMEA and considering only the comparison between the configurations that operate with the classic tree-based GP algorithm, GP-GOMEA is the engine that achieves the best solution to the problem. Regarding the time required for the evaluation process, ECJ-GP reaches the lowest time. Concerning the time needed to validate the best expression in a set of unseen instances, no fair comparison can be made since this time varies with the size of the best-evolved policy. Therefore, since the best expression found with the TensorGP algorithms are the ones with the smallest size, these are the executions which require less time to be computed. Also, with TensorGP, there is a slightly lower value for GPU execution, both in training and validation time. Additionally, considering a situation in which the algorithms in the different programming languages found the same best policy, it is expected that C++ is the one that can manage the validation process more quickly.

Considering the focus of this dissertation, the time that will impact the success of the implementation is the validation time. In a real-world application, it is crucial to quickly construct the routes of a working day. This is done by applying a routing policy to a problem instance, which corresponds to the validation phase of the developed approach.

After analysing all results of these experiments, a decision must be made regarding the learning policy algorithm to employ. Considering both the achieved routing policy and validation time, GP-GOMEA is the chosen engine since it provides a good trade-off between these two criteria and has the additional feature GOMEA, which can improve further the performance of the tree-based GP approach.

4.2 Literature Instances

Now that the GP algorithm is selected, we can proceed with the experiments on the actual problem, the DVRPTW. The computational experiments were conducted on several DVRPTW instances from the literature. Two different benchmark instance groups were considered, and both of them extended the classic Solomon VRPTW instances [16].

The Solomon instances are composed of six different types of problems, named R1, R2, C1, C2, RC1 and RC2, where each data set holds between eight to twelve 100-node problems. Each instance type has the following meaning: type C has clustered customers; type R has customers with uniformly randomly generated locations; type RC has a combination of clustered and randomly positioned instances. Also, the planning horizon can be short (type 1) or long (type 2), allowing fewer customers per vehicle's route but more used vehicles or fewer routes used but more populated. Considering the VRPTW, each problem instance includes the following features: number of vehicles K, the capacity of each vehicle Q, number of customers V', and, for each customer comprises: location (x_i, y_i), demand d_i, lower l_i and upper u_i limits of the time window and service duration s_i. The travel time between two vertices is typically assumed to be equal to the travel distance, calculated by the euclidean distance between vertices. Additionally, each 100 customer instance can be easily converted into smaller instances by considering only the first 25 or 50 customers.

The first benchmark instance group was adapted from the instances presented by Necula, Breaban and Raschip [75]. The authors added a new feature to the Solomon instances in order to extend the static version of the problem into the DVRPTW. To this end, the newly added feature represents the time each customer becomes available to the system. This attribute was added to each instance based on a dynamicity level X%, representing the percentage of dynamic customers. This value X is calculated with the metric shown in Equation 2.1, proposed by Lund et al. [61]. Higher percentages for this level means that more dynamic customers are included, whilst 0% indicates only static customers are presented. The authors generated instances of the six different problem types. Two problem instances were chosen for each category, and for each instance, three values of dynamicity were selected: 10%, 50% and 100%, giving a total of 48 DVRPTW instances. This set of instances was used for both preliminary and final experiments conducted, but some were converted into 25 and 50-node instances to increase samples' variability.

The second group of instances was added to the set of instances of the final experiments to increase the size of the set . These instances were proposed by Lackner [76] and follow the same paradigm as the instances of the first group. One extra feature with the available time was added to the Solomon instances, and each instance has a degree of dynamism. In this case, dynamicity levels of 10%, 30%, 50%, 70% and 90% are represented, and each instance has 100 customers. Considering these five degrees of dynamism and all the 56 base Solomon instances, a total of 280 instances is achieved. For experiments' execution, this group of instances was reduced to 232 since some were identical to instances of the first group.

The entire group of DVRPTW instances is divided into two different instance sets: training

and validation. The training set is used in the evolution process of the GP algorithm, and the validation set is used to evaluate the best expressions in never seen instances. For the training phase, 20% instances were considered, resulting in the remaining 80% for validation.

4.3 Experiments Design and Parameters

The use of GP involves tunning many parameters, such as population size, number of generations, selection type and reproduction, crossover and mutation rates. This task is crucial for obtaining a good solution to the problem, being related to possible complications that may emerge related to premature convergence.

This section describes all the preliminary experiments that were carried out to find the best combination for the algorithm parameters to be used in the runs (Section 4.3.1). Further, the different configurations tested for the GP terminals and functions are detailed (Section 4.3.2). For the execution of all the preliminary tests, the GP-GOMEA algorithm is employed, with the GOMEA feature turned off, resulting in the classic tree-based GP algorithm, which will be called GP-tree from now on. The structure of this algorithm is detailed in Section 3.2.2, Algorithm 3. Regarding the problem instances, the first group of instances explained above were used for these preliminary experiments with a rate of 60% and 40% for train and validation, respectively.

4.3.1 Algorithm Parameters

The preliminary experiments phase for tunning the algorithm parameters was organised in different steps, one for each group of parameters. In each step, the choice is made based on the trade-off between performance and size. The performance of a policy is calculated using the fitness metric for training described in Section 3.2.1. In Equation 3.12 the value for α is set to 0.245 and β is set to 27.182. These costs were calculated based on the case study's real costs since no studies were found where these values were specified. The calculation of these constants will be detailed in Chapter 5. Five independent runs were performed for each configuration of the parameters.

First, an experiment was performed to evaluate an adequate value for the population size. Three different values were considered: 50, 200 and 500. Table 4.6 shows the results of executing each of the values. The *min fitness* column represents the lowest overall fitness value of the best-evolved individuals for the five independent runs. While the *average fitness* column denotes the average of all fitness values of the best-evolved individuals. It is also included a column for the *average size* of the best-evolved policies. Cells in bold represent the best performance achieved.

Table 4.6: Performance and size of policies for different values of population size.

Population Size	Min fitness	Avg fitness	Avg size
50	163525.0	166709.4	24.2
200	151357.0	156474.0	26.2
500	**141364.0**	**151674.6**	28.6

It can be observed that larger population sizes result in better results. Also, best fitness values generally mean larger sizes for the individuals. From this experiment test, a population of 500 individuals was selected since it provides the best trade-off between performance and size. One should note that larger values for this parameter could improve even more the obtained results, but, at some point, the improvement in solution quality may be negligible. In either case, there are limited resources available.

The second test performed concerns the GP operators, namely, tournament size, reproduction rate, crossover rate and mutation rate. Three different configurations were considered for this experiment. The first one considers the crossover rate higher than the mutation rate and a tournament size of 7, which is usually the assumption made in the literature (e.g. used in [23–26]). However, the exact opposite configuration is examined in the third configuration regarding the mutation rate with a higher value and a tournament size of 2. This design is also used in some GP studies in the literature, e.g. in [6]. For the second configuration, a middle case is studied with equal rates for crossover and mutation and an intermediate value for the tournament size.

Table 4.7: Performance and size of policies for different configurations of tournament size, and reproduction, crossover and mutation rates.

	Crossover	**Mutation**	**Reproduction**	**Tournament**	**Min fitness**	**Avg fitness**	**Avg size**
Config 1	0.80	0.15	0.05	7	151357.0	**155029.8**	26.2
Config 2	0.475	0.475	0.05	4	153431.0	158929.4	28.2
Config 3	0.15	0.80	0.05	2	**149393.0**	157988.4	28.2

Table 4.7 presents the results of the test. From the analysis of the table, it is possible to verify that no significant changes occur when varying the rates of GP operators. Configuration 1 achieves the minimum average fitness, but Configuration 3 achieves the minimum overall fitness. From this experiment, no clear conclusion can be made regarding the most favourable combination of parameters. Therefore, it may be useful to analyse more deeper the expressions generated by GP with these different configurations.

Since this dissertation focuses on finding both interpretable and high-performing expressions, it is helpful to encounter the concept of dominance. A solution is said to be non-dominated or Pareto optimal if no objective function's value can be enhanced without affecting some other objective values. In this case, an expression of size s, produced by algorithm A, is said to be non-dominant if, for all expressions generated by A, there are no expressions of equal or smaller size which achieved a lower fitness value in the training set than the expression of size s.

Figure 4.2 represents the non-dominated expressions and the best-expressions found for each experimental run. Here it is possible to verify that Configuration 1 permits achieving the solutions with the best trade-off performance and size for almost all expression sizes. The only exceptions are noticed in expressions of sizes 1 and 29, which, surprisingly, both result from the use of Configuration 3. Given the results of this test, Configuration 1 was chosen for the GP operators parameters.

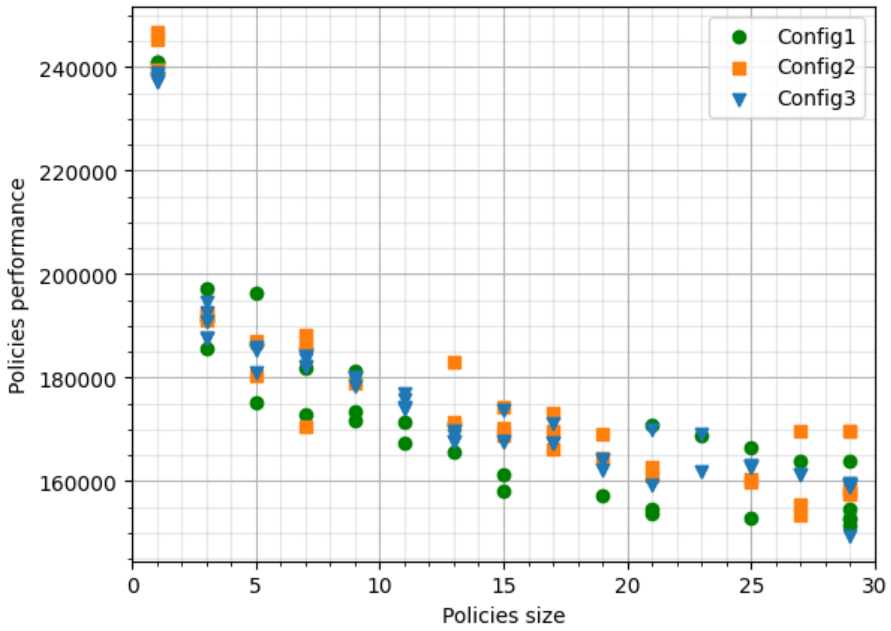


Figure 4.2: Performance and size of policies for different configurations of GP operators.

Remembering that we are focused on enabling the interpretability of the evolved routing policies, GP was explicitly constrained to generate solutions with trees of relatively small size. Accordingly, the parameter maximum solution size was set to 29 nodes. This value was chosen after analysing the work of Virgolin et al. [77], in which the authors recommend using a limit less or equal to 31 nodes, affirming that sometimes it is even non-trivial to interpret solutions of this level. Also, in this article, interpretability is seen as a concept influenced by multiple factors, including the size and complexity of the set of terminals and functions that compose the GP solution. The parameters of these sets will be explored in the following section, Section 4.3.2.

Concerning the other hyper-parameters that need to be specified before the algorithm's execution, conclusions were drawn with the realisation of the previous tests. The number of generations was defined as 50 since, in general, no fitness function improvement is observed beyond generation 40. In what relates to the height of the tree, it was noted that prematurely limiting it too small could lead to the prune of good branches, so a commonly used value for the maximum tree height was set (17). For the population initialisation a maximum tree depth of 5 was used and the method chosen was RHH. Table 4.8 shows the GP parameters that will be used in the remaining experiments.

4.3.2 Function and Terminal Sets

The GP algorithm uses a set of terminals and functions to design the routing policy, which, given the fitness function, undergoes an evolutionary process to learn how to construct better solutions. The definition of the set of terminals and functions used by GP comprises a key decision in the

Table 4.8: Final list of GP Parameters.

Parameter	**Value**
Maximum generations	50
Population size	500
Tournament size	7
Elitism	10
Crossover rate	0.85
Mutation rate	0.15
Reproduction rate	0.05
Initial generation method	RHH
Initial maximum tree height	5
Maximum tree height	17

solution approach since they will dictate which components can be used in solution representation and how they can be combined. The size of these sets has also a huge impact on the size of the solution space, which grows exponentially, and hence it will also determine how well that space can be explored. The selection of those sets depends on human intelligence, involving a deep analysis of the problem to extract relevant variables and constraints. Also, it is important to note that the terminal and function set selection should satisfy the closure and sufficiency requirements. Next, different variants for each set are examined to inquire which ones result in higher quality solutions for the problem. Our implementation does not include ephemeral random constants to promote explainability and generalisability.

Regarding the set of functions, two iterations of the procedure presented in Figure 3.1 were executed, resulting in two GP variants differing in the function set. The first one (GP-Fsmall) includes a smaller list of operators, representing the basic mathematical operations, while the second (GP-Flarge) also includes *min* and *max* operators, based on works found in the literature. The protected division operator returns one if the denominator is zero. The function sets examined for each variant are shown in Table 4.9.

Table 4.9: Complete list of considered functions.

Function	**Description**	**GP-Fsmall**	**GP-Flarge**
+	Addition	x	x
-	Subtraction	x	x
*	Multiplication	x	x
p/	Protected Division	x	x
min	Minimum		x
max	Maximum		x

Five independent runs were executed for each configuration. Table 4.10 shows the results achieved, both in fitness values and solutions' size. The incorporation of the *max* and *min* operators in the function set resulted in poorer solutions performance accompanied by slightly larger

expressions.

Additionally, Figure 4.3 shows the best expression found in each independent run as well as all the non-dominated expressions of each run. It is possible to verify that, for smaller sizes of the policies (1-7 nodes), the non-dominated policies overall presented policies are achieved by the GP-Flarge algorithm. However, as the size of the policies increases (>7 nodes), the variant that finds the Pareto optimal policies is the GP-Fsmall. Hence, and considering that the GP-Fsmall variant achieves the lowest average and minimum fitness values, the mathematical operators considered in it will be selected for the remaining experiments.

Table 4.10: Performance and size of policies for different configurations of GP functions.

	Min fitness	**Avg fitness**	**Avg size**
GP-Fsmall	**151357.0**	**155029.8**	27.4
GP-Flarge	157763.0	159954.0	26.1

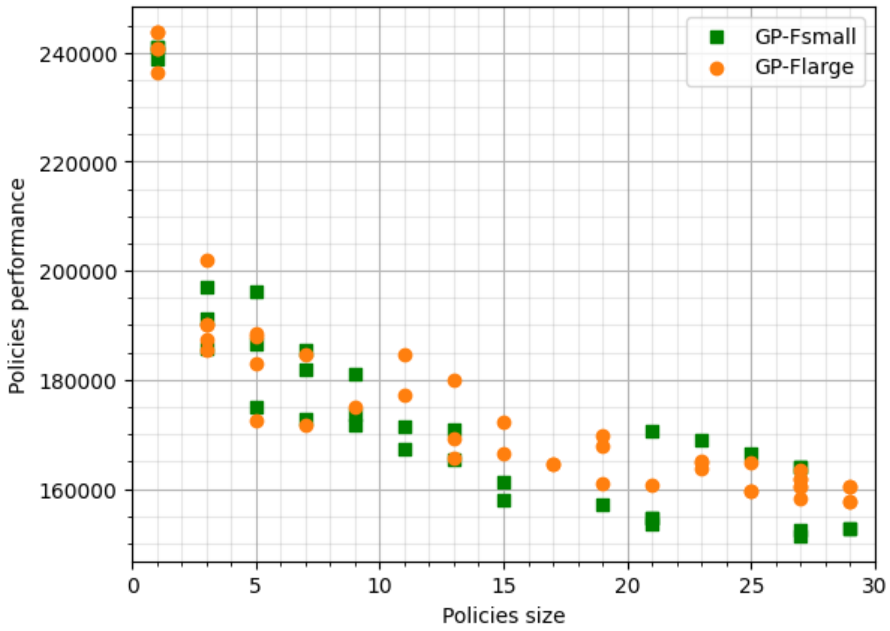


Figure 4.3: Performance and size of policies for different configurations of GP functions.

Concerning the selection of the set of terminals, three iterations of the approach presented in Figure 3.1 were executed, resulting in three GP variants differing in the terminal set. Table 4.11 presents the list of terminals considered for each iteration. Generally, the terminals under consideration are similar to the traditional ones already employed in the construction of VRP solutions. The algorithm of the first iteration (GP-Tsmall) uses a more straightforward set of terminals, while the second algorithm (GP-Tmedium) adds to the first one more complex features. These features were selected based on a direct analysis of the problem and extracting the parameters that condition it. Also, the few studies that employ GP for VRP ([23, 24]) were analysed, and the terminals

used there were extracted if adequated. For the third variant (GP-Tlarge), existing studies in the literature that apply GP to other combinatorial problems (e.g. [6, 25, 54]) were studied to inquire if the terminals used there could be adapted for our problem. The extracted terminals were added to the terminal set of the third variant, resulting in a large terminal set with 16 variables.

One should note that the function and terminal set size can affect the evolution, and the size of the terminal set in the GP-Tlarge algorithm may be unnecessarily large. Also, the fewer features are involved in the routing policies, the easier it will be to interpret them since perhaps simpler combinations between these features will stand. For instance, in [6], the authors verified that smaller sets of terminals and functions evolved higher performance policies than more extensive sets of these symbols.

Table 4.11: Complete list of considered terminals.

Terminal	**Description**	**GP-Tsmall**	**GP-Tmedium**	**GP-Tlarge**
STW	Start of the time window	x	x	x
ETW	End of the time window	x	x	x
D	Travel distance	x	x	x
T	Travel time	x	x	x
S	Slack time	x	x	x
ST	Service time		x	x
RT	Return time to the depot		x	x
RC	Vehicle's remain capacity		x	x
Q	Demand		x	x
AVGT	Average time from remaining nonserved customers to the considered customer		x	x
AVGTL	Average time from remaining nonserved customers to the last customer added to the route		x	x
C	Extra cost of serving the considered customer			x
CR	Critical Ratio			x
RTT	Route total time			x
NRTT	Route total time after adding the consider customer			x
U	Route utilization			x

Each GP variant was executed in five independent runs. Table 4.12 shows the average and best performance of the best-evolved policies as well as the average policies' size. Results show that although GP-Tlarge utilises a more extensive terminal set, it is the variant that permits achieving the best overall performance while maintaining an acceptable size for the evolved trees. Also, there is no significant difference in size across the different GP variants.

The chart in Figure 4.4 presents the best-evolved policy and the non-dominated policies of each independent run. Here it is possible to observe that GP-Tlarge for all range of expression sizes shown is the GP variant which achieves the best expressions.

The next step is to evaluate the frequency of appearance of each terminal in the GP-Tlarge' set

Table 4.12: Performance and size of policies for different configurations of GP terminals.

	Min fitness	Avg fitness	Avg size
GP-Tsmall	159234.0	161356.4	25.8
GP-Tmedium	151357.0	155029.8	26.2
GP-Tlarge	**141645.0**	**145308.2**	25.4

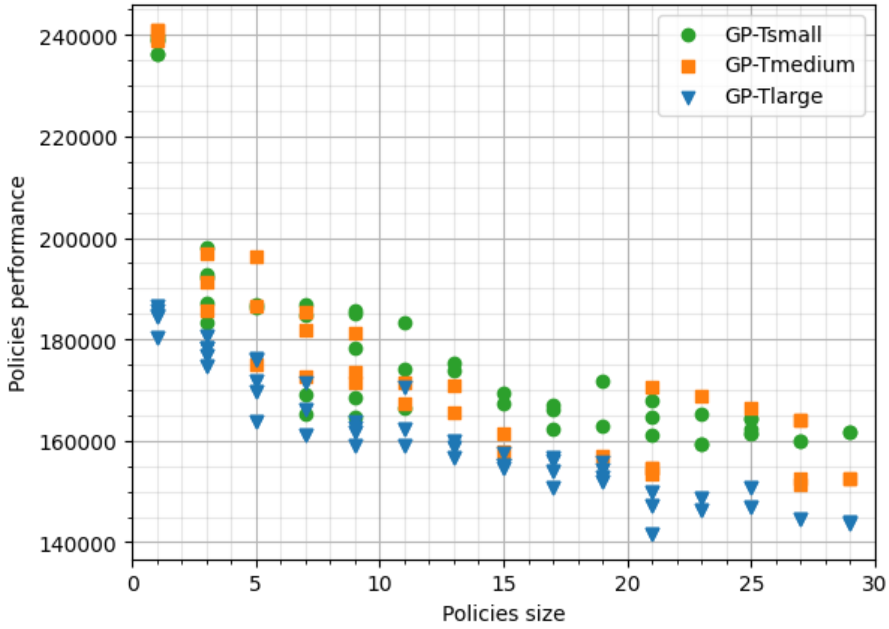


Figure 4.4: Performance and size of policies for different configurations of GP terminals.

to investigate if there are some terminals worsening the performance of the evolved policies. The aim is to identify those terminals and trim the set to a smaller size. Figure 4.5 shows how often the symbols appeared on average in the best x% expressions (of the final population) in the five independent runs for the GP-Tlarge variant.

It is possible to observe that the terminals "U" and "S" never appeared in the best 1% and 10% individuals and barely appeared in the 50% and 100% best ones. On the other hand, "C", "AVGT", and "T" are always the more relevant symbols for the evolutionary process, each covering more than 10% of the appeared symbols in all evolved individuals.

In order to eliminate some parameters from the terminal set, a criterion was established which considers that if a terminal appears in less than 5% of the 100% best-evolved individuals, then it should be discarded. Thus, considering this condition and observing the line drawn in Figure 4.5, which represents this threshold, the following terminals were discarded for the final set of terminals: "STW", "ETW", "S", "ST", "Q", "RC" and "U".

Table 4.13 presents the final list of terminals and functions considered for the final experiments.

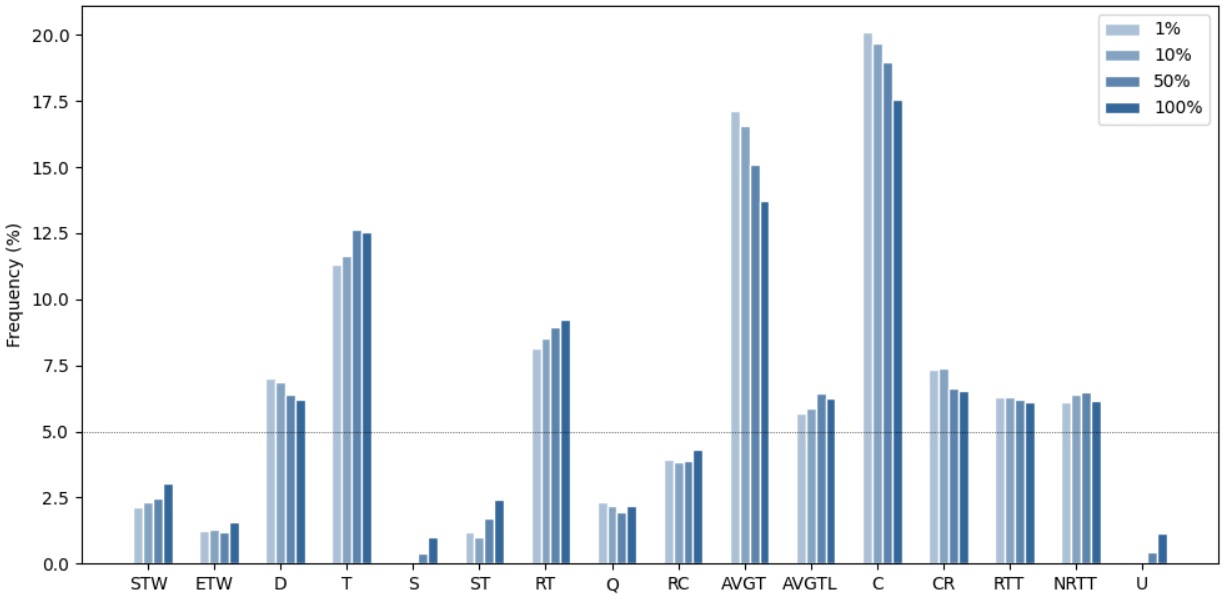


Figure 4.5: Frequency of terminals in the top x% policies evolved.

Table 4.13: Final list of GP functions and terminals.

Parameter	**Description**
Functions	
+	Addition
-	Subtraction
*	Multiplication
p/	Protected Division
Terminals	
D	Travel distance
T	Travel time
RT	Return time to the depot
AVGT	Average time from remaining nonserved customers to the considered customer
AVGTL	Average time from remaining nonserved customers to the last customer added to the route
C	Extra cost of serving the considered customer
CR	Critical Ratio
RTT	Route total time
NRTT	Route total time after adding the consider customer

4.4 Obtained Policies

In this section, the best-evolved routing policies will be exhibited and analysed. Following the work described in the previous sections, the best-found set of hyper-parameters was used, and the GP approach was executed 10 times. The computational tests were performed on AMD Ryzen Threadripper PRO processing units running at 4.2GHz, within a Ubuntu 20.04 virtual environment with 8 CPUs and 44GB, each test executed with two threads in parallel.

The quality of the solutions will be evaluated on the validation set with instances from the literature. The objective is to find equally effective and interpretable routing policies which can quickly produce solutions for the problem instances. These objectives will be measured base on the relative performance of the policies in the validation set, their size and the average time required to evaluate an instance of the problem.

For the execution of these tests, two different GP algorithms will be considered: the GP-Tree algorithm, which has been used so far, and the GP-GOMEA algorithm, which adds to the first one the GOMEA feature. As analysed before, combining GOMEA with GP brings multiple advantages to the evolutionary process, including a deeper exploration of the search space and a decreased susceptibility to bloat.

On the other hand, due to the high number of evaluations performed, using GOMEA in the GP evolutionary process results in a high running time. Additionally, as stated in [46], GP-GOMEA typically requires much smaller populations than GP-Tree due to the high number of mixing trials executed by the GOM operator. Therefore, and since limited time resources are available for the experiments' execution, the GP-GOMEA algorithm will be employed with a reduction in the size of the population to 100 individuals. Concerning the representation of the solutions, a difference is applied when the GOMEA feature is used. Contrary to the GP-Tree algorithm in which solutions can have any shape, in GP-GOMEA, they have a fixed structure with the maximum number of nodes allowed for a given tree height and functions arity. In this analysis, all functions have arity two, and the initial tree height is set to 5, so the policies in GP-GOMEA can have up to 63 nodes. Yet, some nodes are introns, meaning that they are ignored in the computation of the output, occurring in cases where the terminals are not in the maximum depth. Hence, the maximum solution size parameter defined previously for the GP-Tree algorithm is ignored when using the GOMEA feature. The remaining GP parameters and the list of terminals and functions, presented in Tables 4.8 and 4.13 will be maintained for the GP-GOMEA. Regarding the GP-Tree algorithm, all the GP parameters and the list of terminals and functions are preserved.

Figure 4.6 illustrates the evolution of the fitness of the best-performing individual in the population throughout the generations for all the ten independent runs executed. Here, the capability of GP-GOMEA in exploring the search space is evident. It converges much faster than GP-Tree to higher-performing solutions. In the first generation, GP-GOMEA's fittest individual is, on average, four times more performative than the best-performing individual of the GP-Tree algorithm, which achieves the same value only around generation 7 after an abrupt decline in the fitness value. Besides, the first generations of the evolutionary process comprise the most significant improve-

ments in individuals' fitness, with minor changes in the remaining generations. At the end of the evolutionary process, when the stop criterion is triggered, the GP-GOMEA algorithm tends to find solutions with lower (better) fitness values.

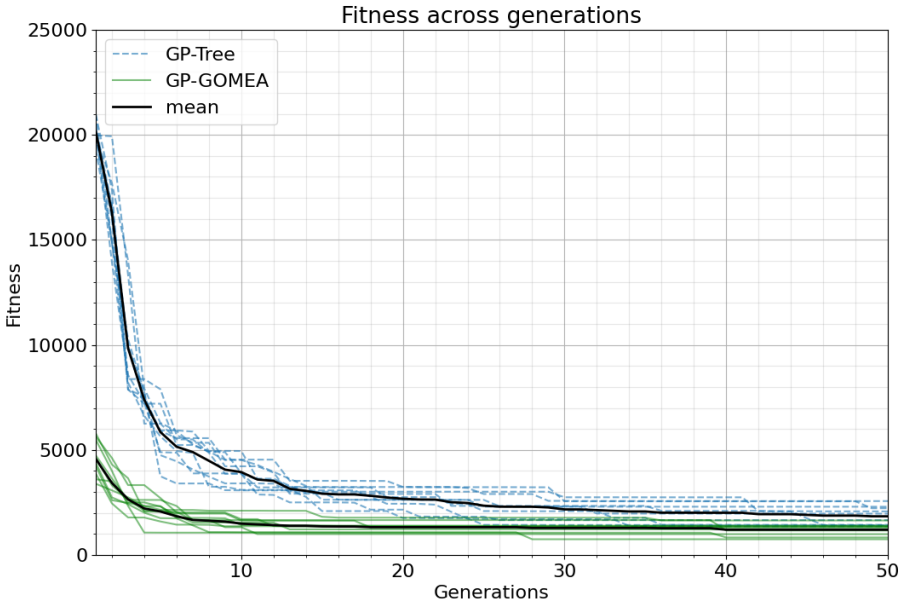


Figure 4.6: Fitness evolution through generations.

In Figure 4.7, all the non-dominated policies of each independent run, as well as the best performing policy of each run are represented. The fitness values correspond to the values achieved in the training process. Figure 4.8 zooms the y axis to enhance the analysis of expressions with lower fitness values. Here, the best expressions fitness for sizes 1,3, 5 and 7 are not displayed since, compared to the other expressions' sizes, they present much worse performance.

Figure 4.8 points out eight non-dominated policies (RP1 - RP8). Among them, five in eight policies were evolved with the GP-Tree. However, the ones with the lowest train fitness values were generated with the GP-GOMEA algorithm (RP7 and RP8).

Figure 4.9 shows the relationship between the size and the validation performance of the eight best non-dominated training policies. Also, Table 4.14 gives more details of these policies, presenting their mathematical expression, train and test fitness, size and the algorithm which evolved them.

Firstly, it is important to note the significant increase in values on the y axis. This increase in the expressions fitness value is due to a higher number of delays in the instances evaluated. Since the delay is penalised by a constant of a very high value (M), a slight increase in this variable results in a high increase in the fitness value of the expression.

Secondly, it is evident that these policies do not follow the same pattern in the validation set as in the training set, i.e., not all non-dominated policies in the training set remain non-dominated

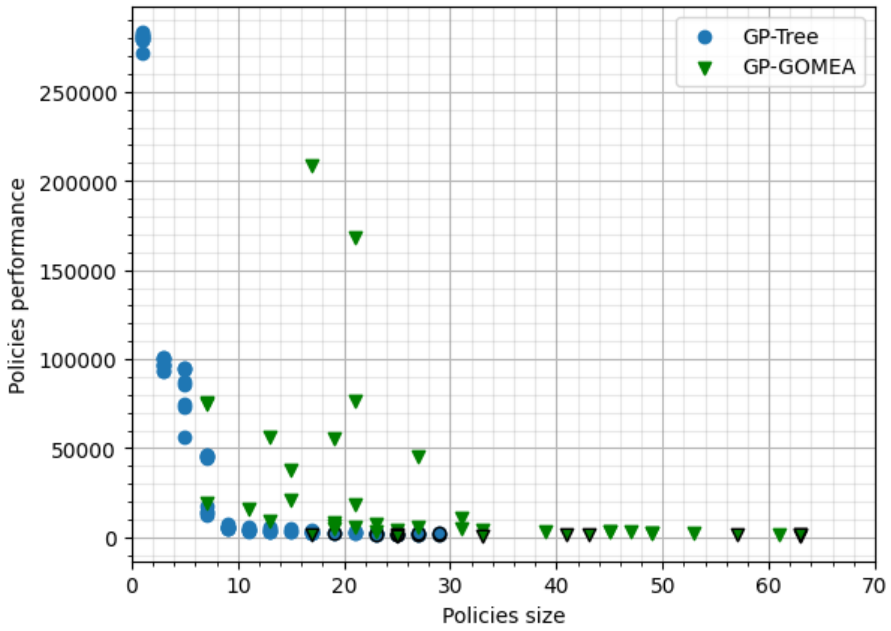


Figure 4.7: Performance and size of the best and non-dominated policies of the 10 runs in the training set. The points circled in black represent the best-evolved policy of each run.

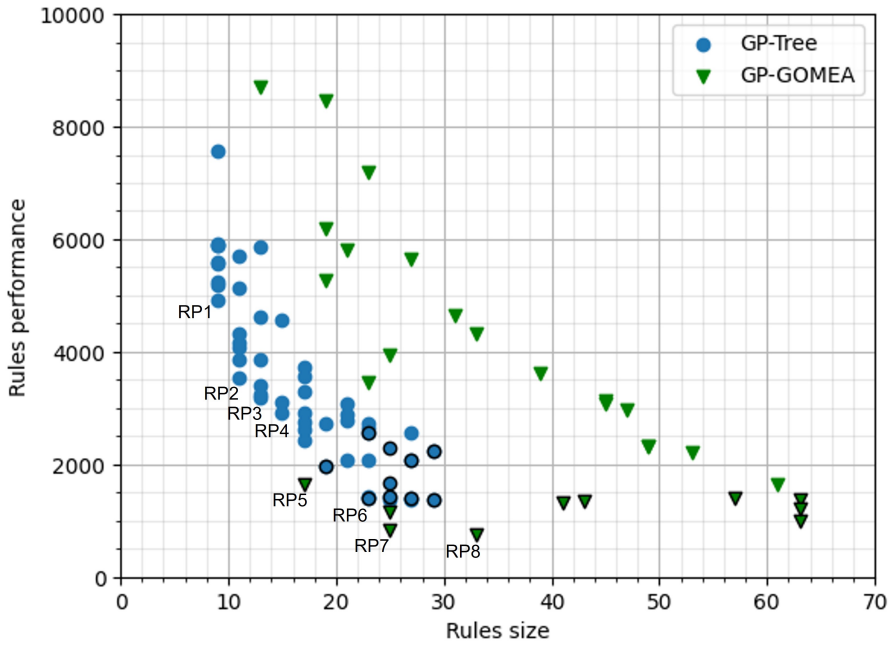


Figure 4.8: Performance and size of the best and non-dominated policies with fitness value lower than 10000 of the 10 runs in the training set. The points with black borders represent the best-evolved policy of each run.

in the validation set. This inconsistency is possibly due to the presence of overfitting, especially in policies RP4 and RP6. Unfortunately, although the train and the validation instances present similar internal characteristics, the model may be too close to the training instances. Yet, another reason for this event, and probably the most accurate one, may be an imbalance between the two sets. The high number of literature instances collected led to a division of 20% and 80%, respectively, for the train and validation set so as not to weigh the training phase too much computationally.

Finally, it can be observe that the RP7, which was, along with RP8, the policy with the best performance in the training set, is the most performing policy in the validation set. Accordingly, since it provides an acceptable trade-off between size and performance, policy RP7 will be selected as the best-found policy using literature instances.

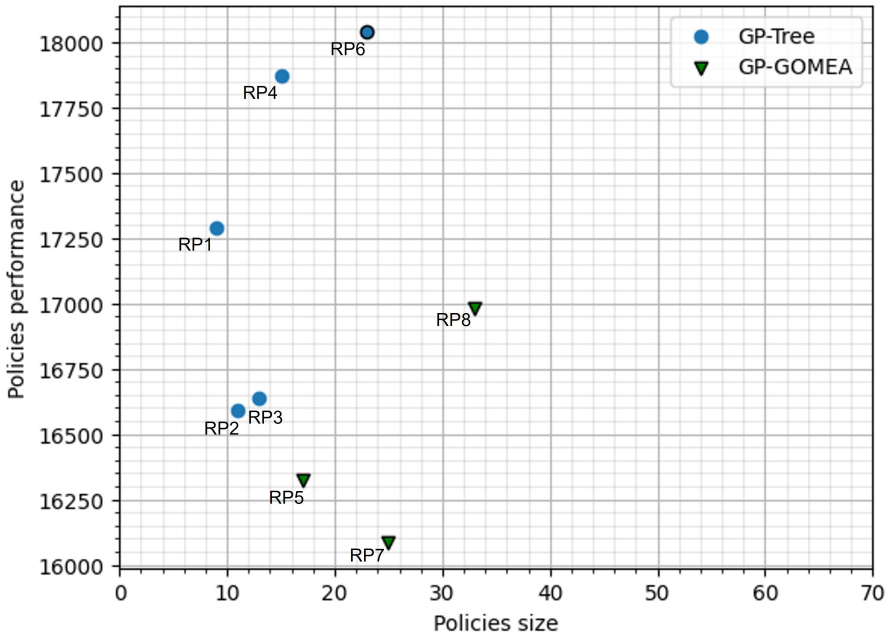


Figure 4.9: Performance and size of the best non-dominated policies in the validation set. The points circled in black represent the best-evolved policy of each run.

Table 4.14: Best non-dominated routing policies in the validation set for the literature instances.

Rule	Mathematical Expression	Train Fitness	Test Fitness	Size	Algorithm
RP1	$(RT*NRTT)p/(CRp/NRTT-C)$	4929.98	17291.90	9	GP-Tree
RP2	$((CR-NRTT)*NRTT)p/(C*(C+CR))$	3539.30	16591.15	11	GP-Tree
RP3	$(D*RT)p/((Dp/(RT+CR*D))-C)$	3175.63	16640.74	13	GP-Tree
RP4	$((CR-NRTT)*NRTT)p/(((CR+C)*NRTT)*(C+C))$	2908.41	17870.66	15	GP-Tree
RP5	$((((RTTp/CR)p/(RT-RTT))p/((NRTT+AVGT)*D))-NRTT)+C$	1643.71	16322.03	17	GP-GOMEA
RP6	(NRTT*AVGT-(CR-RT))p/(((CR-C)p/((RTp/AVGTL)+ (CRp/CR)))-(C+RTT))	1408.59	18041.51	23	GP-Tree
RP7	$(T*RT)p/(((Dp/(CR+T+AVGT+CR+CR*T))p/(C+CR*T))-C)$	835.56	**16085.83**	25	GP-GOMEA
RP8	RT+((((RTp/CR)-(RT+T))p/((RTT*C)-(NRTTp/RT)))+ (((T-RT)p/(D+AVGTL))+ ((C+D)p/(NRTTp/C))))	749.06	17981.07	33	GP-GOMEA

Chapter 5

Case Study

The evaluation of the obtained policies in a wide range of instances from the literature attested their performance and provides comparable results for further research. However, in this chapter real-world data from a large European retailer is used, in order to validate the added value in practice. The developed approach was validated with real-world data from a large European retailer. This chapter will present an overview of the case study (Section 5.1), the methodology employed to collect the data and generate the real instances (Section 5.2), and an analysis of the results obtained (Section 5.3).

5.1 Context

The company targeted by the case study is a large European food retailer and has an e-commerce operation. Over the last few years, the online retail sector has gained a significant boost due to the advance of the internet and the continuous digitalisation of modern life. As a result, achieving efficient logistics while maintaining customer satisfaction is of utmost importance.

Currently, in the online retailer considered, inefficiencies exist in the operation of the e-commerce, related to a non-uniform distribution of customer deliveries across the working day. This is due to customer preferences which are clustered around the same delivery window, resulting in high pressure in the transport and logistics systems during some slots, commonly at the end of the day. Therefore, there is a project to develop an approach capable of evening out this distribution, benefiting retailer operational efficiency.

The approach developed within the project will entangle a dynamic time slot management system in which time slot prices vary with each time slot's current operational load and the geographical distribution of the requests. In the current time slot management system of the company, the prices of the time slots for the client to choose their goods to be delivered vary but are only dependent on the slot hour of the day. However, a more robust system in which the clients could be incentivised to choose slots not too crowded could be beneficial. Figure 5.1 presents an overview of the approach developed within the project.

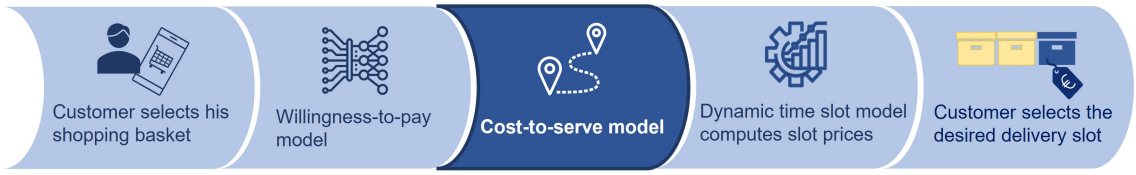


Figure 5.1: Case study approach overview.

A symbolic-based algorithm will be employed to generate an expression that defines time slots and pricing, while yielding the explainability and speed required by these operational decisions. The speed of generating the prices for the slots is a crucial component since the company does not want the customer to be waiting to finish his purchase.

The symbolic expression will specify the price of each time window a customer is presented with, weighing customers' preferences, the cost to include it in the current routes and serve it, the expected demand for the slot and any relevant logistics constraints. Accordingly, before the computation of the dynamic time slot model capable of generating this symbolic expression, two preliminary models must be developed to reckon the expression's parcels. At first, a willingness-to-pay model will predict how much a customer is willing to pay to have his order arrive within a specified delivery slot. Simultaneously, a cost-to-serve model will estimate the cost of serving a customer within a given slot. Finally, the dynamic time slot model is able to compute the price of each time window a customer is presented with.

The work developed in this dissertation can be applied to the cost-to-serve model to satisfy the necessity of quickly estimating the cost of serving the customer. The symbolic expressions generated in the scope of this dissertation can be used as a feature of the time management model to dictate the expected cost of serving a customer, helping in the process of choosing slots pricing. Additionally, since this dissertation employs GP, a symbolic-based approach, it should be possible to achieve both the global explainability and the speed required by the project's approach.

The operations of the considered retailer are conducted in different warehouses, which operate independently in terms of distribution. In addition, the places where the company's fleets are localised usually also function as warehouses and stores. Therefore these concepts will be treated equally in the description and implementation that follows.

5.2 Real Instances

The developed model and solution approach are tested with real-world data from the case study's company. In order to do that, instances of the DVRPTW were created using real historical data from the company's operations. Instances for different warehouses were considered during a certain time interval of operations.

The historical data made available was organized in different tables, with information regarding the customer orders along that year, characteristics of the stores, the precise location of the nodes and the routes performed. All data was relative to the operations in the year 2017.

Two stores were selected for instances' generation. The idea was to choose two stores whose operations were focused on distinct areas and whose characteristics (such as the average number of requests per day or the operating radius) differed. The documented data was relative to operations of the retailer across all country. However, in some areas, the data was sometimes incomplete and, therefore, could not be chosen to generate the instances. Table 5.1 shows the characteristics of the stores chosen. It is possible to note that the level of dynamicity has, on average, low values for both stores considered.

Table 5.1: Characteristics of the stores chosen.

	Store 1	**Store 2**
Average number of requests per day	90	325
Operating radius (km)	181.9	252.8
Average number of needed vehicles	15	65
Average dynamicity level of requests (%)	3.0	4.2

Regarding the time interval selected for each store, due to data limitations, one month of operations was selected for store 1 while, for store 2, only one week was selected. After selecting the depots and the time interval to operate in, the next step was to collect all relevant data and organize them into instances. Considering that each instance represents a day of operation, that two depots were picked and that the time interval represents a month of operations, 39 real instances were created. The aim was to generate instances with an equal format as the non-real instances described in Section 4.2. Table 5.2 shows the parameters of the instances and the way they were estimated with real data from the company. In some parameters, approximations were made for the value since not all information was clearly shown in the data. This was the case of the following parameters: the number of vehicles $|K|$, the vehicles capacity Q, the length of the working day T and the cutomer service time s_i.

Finally, the real instances were created after gathering all the necessary data. Nevertheless, it is necessary to extract the distances and travel times between all vertices before using the real instances in the DVRPTW simulation model. In the non-real instances, these values are calculated through the Euclidean distance. However, since, in this case, real-world problems are considered, a more accurate procedure is required for these calculations. Therefore, considering that these measures were not provided in the historical data made available, an open-source vehicle routing visualisation package was used. This python package called VeRoViz [78] provides a wide range of features, including a function that captures matrices for ground-based travel time and distance between nodes. This function was used in our implementation, and the fattest travel time and travel distance between all nodes were extracted.

Additionally, before passing for the program execution with the real instances, it was necessary

Table 5.2: Instance parameters and real data collected.

Parameter	**Description**	**Real data collected**
|V|	Number of customers	Number of orders
|K|	Number of vehicles	Average number of vehicles used in the routes
Q	Vehicle capacity	Maximum number of boxes in a vehicle in the routes
x_0, y_0	Depot location	Store location
$T \quad \forall i \in V$	Length of the working day (depot time window)	Difference between the maximum arrival time and the minimum departure time at the depot in the routes
$x_i, y_i \quad \forall i \in V$	Customer location	Order location
$l_i, u_i \quad \forall i \in V$	Lower and upper limit of a customer time window	Slot chosen by the customer
$s_i \quad \forall i \in V$	Customer service time	Average time stayed in the location in the routes
$a_i \quad \forall i, j \in V$	Customer available time	Time where the order was made by the customer

to estimate the cost for the company of travelling per distance unit (α) and of rejecting a customer (β), parameters of the objective function (Equation 3.1). Regarding constant α, this value was directly extracted from a table with statistical data from the company. Concerning the value of β, it was considered equal to the average profit that the company has with a customer. This value was calculated by averaging the total price paid by the customers in their orders during the time period and in the stores chosen and considering a profit margin of 25% for the company. As a result, α has a value of 0.245 and β has a value of 27.182. As stated previously, the values of these constants from the objective function were also considered for the experiments with the instances from the literature since no studies were found where these values were specified.

5.3 Results

This section provides the results from using real-world data from the case study to train and validate the routing policies evolved in the GP algorithm. Additionally, the obtained results are compared to the real routes performed by the company in the same planning horizon and to manually designed heuristics commonly used for dynamic transportation problems.

Currently, the company utilises a commercial software system to compute the routes, which is very different from the approach developed in this dissertation. In contrast to a dynamic transportation problem in which each time a new client arrives, a solution to the problem is computed, the currently employed system is only computed for a group of static customers. Therefore, the company achieves a perfect information solution since the routes are generated only when all requests' information is known, i.e. after the cut-off hours.

Regarding the execution of the system developed in this dissertation in a real scenario, it was performed with a modification in the DVRPTW simulation model. Time slices were eliminated,

and now a solution to the problem is computed each time a new customer request arrives. Although the use of time slices in solution construction can help reduce the number of times a solution is computed, the aim here is to consider the most realistic scenario possible in the simulation model computation.

Concerning the group of real instances used in the execution, it was divided between training and validation sets, with a rate of 20% and 80%, respectively. The values presented in Table 4.8 were used as input parameters and the symbols presented in Table 4.13 were considered for the terminal and function sets. The computational tests were performed on AMD Ryzen Threadripper PRO processing units running at 4.2GHz, within a Ubuntu 20.04 virtual environment with 8 CPUs and 44GB, each test executed with two threads in parallel. Also, 10 independent runs of the algorithm were conducted.

After gathering the results of all independent runs conducted, the obtained routing policies were analysed. Their size and performance in the validation set were compared, and the non-dominated policies with fitness values behind a certain threshold (600) were extracted. The two routing policies extracted (RPA, RPB) are present in Table 5.3, along with their corresponding train fitness, validation fitness and size. Both expressions were achieved with the GP-Tree algorithm.

By analysing these two policies, policy RPB was chosen for a deeper analysis of the results and for the comparison phase that will be presented next since, between the two, it is the routing policy that achieves the best performance in the validation set while having an acceptable number of nodes.

Table 5.3: Best non-dominated policies in the validation set for the real instances.

Policy	**Mathematical Expression**	**Train Fitness**	**Validation Fitness**	**Size**
RPA	(C+D)p/((RT+RT+AVGTL)*((((RT+AVGTL)p/((C+CR)p/(NRTT+T)))+ AVGTL)*NRTT))	237.69	534.74	25
RPB	(C+CR*T)p/(NRTT+T)-RTp/(Cp/(C+Tp/((T*C+CR)p/(NRTT+RT))))	230.34	**195.31**	27

Regarding the routes actual performed by the company in the same considered planning horizon and for the same considered stores, some metrics were calculated to evaluate the routes and allow the comparison with the results generated by applying our GP approach. After extracting the information about the routes from the previous data made available, some metrics were immediately extracted from the data (e.g. number of used vehicles, number of late arrivals and number of postponed customers) and some needed to pass through calculations (e.g. routes total cost, total travelled distance and total travelled time). Concerning the total travelled distance and time of the routes, the matrices used in our implementation, which have the value of the distance and time between all nodes, were utilised to calculate these parameters. After that, the objective function presented in Equation 3.1 was used to calculate the total cost of the routes for each day of work.

Additionally, some heuristics were included in the comparison of the results. The main reason behind it was to consider heuristics that the company would possibly adopt to solve a dynamic version of the problem. Thus, a fairer comparison can be made with our GP approach since, in

this way, we are comparing procedures which operate dynamically, i.e. a solution is computed each time a new dynamic client arrives. This set of heuristics is enumerated and explained bellow:

- **Nearest Neighbour Algorithm (NNA):** always selects the closest customer to the considered node.
- **Cheapest Insertion Heuristic (CIH):** computes the cost between all unserved customers and every customer that is already included in the partial solution. After that, selects the customer who achieves the cheapest insertion among all other unserved customers insertions into the routes.
- **Shortest Return Time heuristic (RT):** always selects the customer whose return time to the depot is the shortest.
- **Shortest Slack time heuristic (S):** always selects the customer whose slack time (time until the end of the time window) is the shortest.
- **Less Remain Capacity heuristic (RC):** always selects the route to add the customer whose remaining capacity is less.

From the set of heuristics presented above, one should note that NNA, RT, S and RC are included in the set of terminals used for GP's evolutionary process. Also, NNA is, in the terminal set, denoted by D.

Finally, having the set of manually designed heuristics, the real routes from the company and the best policy found in our GP approach, the results will be displayed. In addition to this, the best routing policy achieved in Section 4.4 will be evaluated with the real instances from the company. It is important to remember that the policy RP7 was evolved in an evolutionary process trained by literature instances, which are very different from the real instances presented in this chapter. Therefore, it is expected that the results obtained with this policy are not as high-performing as the ones obtained with RPB, which was trained with real instances. Nevertheless, the analysis of the behaviour of the policy RP7 in a set of instances which differs substantially from the ones used to create the model is interesting to inquire how generalisable the solution can be.

Next, a set of tables will present the results of the real routes from the company, the heuristics computation, and the test of policies RP7 and RPB for each company store considered. Each table will compare the results of a metric, being the metrics and its corresponding table the following: total cost of the routes (C), Table 5.4; number of used vehicles (VN), Table 5.5; total travelled distance (TD), Table 5.6; total travelled time (TT), Table 5.7; rate of late arrivals at the customers (LA), Table 5.8 and rate of postponed customers (PC). Regarding the metric PC, no table is presented for its comparison since neither of the solutions obtained with the different models had postponed customers.

Concerning the total cost C presented in Table 5.4, this is arguably the most important metric to analyse since it is a weighting sum of the metrics TD and PC, penalised by the LA parameter.

Also, it is important to balance the results of this metric with VN (Table 5.5) since it is not included in the cost of the routes.

By analysing the presented results, it is possible to verify that GP-RP7, GP-RPB and CIH can all achieve lower costs for the routes than the commercial software adopted by the company. However, only GP-RPB and CIH can achieve this with a number of vehicles lower than the ones in companies' routes. On the other hand, all NNA, RT, S and RC heuristics produce results which are not competitive in terms of costs with the ones generated by the retailer. This remark follows what was expected from these heuristics. Their simple structure only considers a single parameter for constructing the routes, which can hardly be a good generalisation for a complex problem involving multiple constraints.

Also, it is interesting to note that, although the routing policy GP-RP7 was not trained with data from the retailer as the policy GP-RPB, the costs of the routes constructed with it do not present a considerable divergence from the ones produced with GP-RPB. Yet, GP-RP7 uses a considerably higher number of vehicles to produce these costs. Nevertheless, the results produced with GP-RP7 outperform the routes performed by the company and are not far from the ones produced with GP-RPB, representing a favourable point for GP, highlighting its ability to generalise to instances of a problem with significantly different characteristics.

Regarding the CIH, its performance in generating solutions to the problem is notable, producing the minimum overall cost to an instance. However, GP-RPB results are, on average, superior to the ones developed by CIH. Moreover, a crucial point to consider is the time required to compute the routes of a problem instance after adding a new customer. Table 5.9 represents an approximation of the average time needed for this insertion considering a GP policy, the CIH and the NNA, RT, S and RC heuristics. Indeed, since CIH involves a continuous optimisation approach, the time required to insert a dynamic customer into the routes is significantly higher when compared to the time needed to apply a GP policy. Also, comparing the computation time of a GP expression and the NNA, RT, S and RC heuristics, naturally, since the latter represent only expressions with one (NNA, RT, RC) or three (S) nodes, their computation is faster than a GP expression which, in this analysis, includes more than twenty nodes.

In what relates to the group of heuristics which average costs are not satisfactory, S is the one which achieves the best average costs, followed by NNA, RT and RC. The unsatisfactory costs presented by these heuristics, however, are accompanied by a less number of vehicles utilized but, a considerable number of LA (Table 5.8) to the customers. Also, concerning the number of delays, these heuristics are the only ones that presented values for this metric.

The results shown in Table 5.6 related to the TD metric do not add many conclusions to the ones stated before since it is englobed in the C metric. Even so, it can be observed that, unsurprisingly, the routes generated by the NNA are the ones that presented the least total travelled distance. Still, it also triggers many consequences in the LA metric.

Table 5.4: Results of the total cost of the routes (C) for the real instances.

		Real	GP-RP7	GP-RPB	CIH	NNA	RT	S	RC
Store1	Min	111.70	95.74	83.36	**83.02**	100.84	166.08	297.40	370.43
	Avg	249.30	165.71	**144.22**	146.75	298507.36	299391.32	211083.53	361474.35
	Max	433.00	269.96	**232.19**	236.73	780186.00	740262.00	560439.00	900421.00
Store2	Min	582.41	401.71	245.30	**211.66**	30624.00	61177.60	51615.40	31539.70
	Avg	1060.98	623.88	**450.75**	459.79	922365.60	880833.52	517171.06	1600990.00
	Max	1528.47	855.85	**674.51**	749.70	2110210.00	2080700.00	1600990.00	2100960.00

Table 5.5: Results of the number of used vehicles (VN) for the real instances.

		Real	GP-RP7	GP-RPB	CIH	NNA	RT	S	RC
Store1	Min	9	11	7	9	6	**4**	**4**	5
	Avg	14.84	16.68	8.88	13.16	8.32	7.84	**7.28**	8.52
	Max	20	23	13	18	12	11	**10**	13
Store2	Min	32	38	19	26	18	19	**17**	20
	Avg	50.80	52.00	**26.00**	34.40	**26.00**	26.80	26.20	28.80
	Max	61	66	**29**	40	33	33	32	33

Table 5.6: Results of the total travelled distance (TD) of the routes for the real instances.

		Real	GP-RP7	GP-RPB	CIH	NNA	RT	S	RC
Store1	Min	455.90	390.76	340.25	338.86	**232.64**	376.90	511.65	467.06
	Avg	1017.58	676.36	588.66	598.67	**438.06**	781.11	1156.96	1119.75
	Max	1767.34	1101.87	947.70	966.26	**759.99**	1417.39	1793.86	1822.62
Store2	Min	2377.18	1639.62	1001.23	863.91	**710.65**	1922.63	2734.76	2417.14
	Avg	4330.55	2546.44	1839.79	1876.70	**1489.31**	3405.10	4778.79	4599.04
	Max	6238.66	3493.27	2753.12	3059.98	**2547.04**	4806.60	6594.65	6284.31

Regarding the results presented in Table 5.7 related to the metric TT, it can be verified that the real routes performed by the company generally achieve the lowest values. However, no great importance should be given to this aspect since our approach considers a restriction that the company does not consider. It refers to the start of the service in the customer, which we assume should be done after the start of the time window, while the retailer permits its beginning before the time slot opens. Thus, not taking the waiting time into account is reflected in the significant difference in the total travelled times between the approaches.

Table 5.7: Results of the total travelled time (TT) of the routes for the real instances.

		Real	GP-RP7	GP-RPB	CIH	NNA	RT	S	RC
Store1	Min	**1428.25**	6986.51	3853.71	5773.24	4571.43	3718.01	3257.57	4615.45
	Avg	**2646.71**	10270.71	5767.39	8352.54	7184.75	7146.33	5957.82	7936.90
	Max	**3909.87**	14520.20	8065.73	11867.30	10095.80	10213.00	8501.91	12122.00
Store2	Min	**6032.75**	24002.10	14064.10	19475.40	15828.30	17700.70	14962.20	19266.90
	Avg	**9971.71**	32329.50	17885.59	23880.32	22919.98	25112.74	22216.28	27217.46
	Max	**13184.66**	40512.50	20840.00	26718.20	29037.90	30952.90	26966.30	31166.90

Table 5.8: Results of the rate of late arrivals (LA) at the customers for the real instances.

		Real	**GP-RP7**	**GP-RPB**	**CIH**	**NNA**	**RT**	**S**	**RC**
Store1	Min	**0**	**0**	**0**	**0**	0	0	0	0
	Avg	**0**	**0**	**0**	**0**	29.84	29.92	21.08	36.12
	Max	**0**	**0**	**0**	**0**	78	74	56	90
Store2	Min	**0**	**0**	**0**	**0**	3	6	5	3
	Avg	**0**	**0**	**0**	**0**	92.20	88.00	51.60	76.60
	Max	**0**	**0**	**0**	**0**	211	208	160	210

Table 5.9: Approximation of the average time, in milliseconds, required to obtain a solution to the problem after inserting a new customer.

Model	**Avg time to compute solution (ms)**
GP	14.7
CIH	4659.4
NNA, RT, S, RC	0.4

Overall, the results achieved with the routing policy GP-RPB evolved by GP with real instances are notorious. There was a decrease in the costs of 49.2% in average (42.1% for store 1 and 57.5% for store 2) when employing the routing policy evolved by GP. Also, there has been a 43.7% reduction in the number of used vehicles (40.2% for store 1 and 48.8% for store 2).

Next, Figure 5.2 illustrates an example of the routes of a working day constructed with the routing policy GP-RPB. Figure 5.3 shows the routes performed by the company on the same working day. These figures were generated with the help of the VeRoViz [78] python package mentioned above, which offers a function to create a leaflet map, having as input the nodes' location and the routes. The corresponding routes' metrics are presented in each figure's top right corner. It is interesting to note the different sequences of clients followed by the two approaches, which are sometimes the same for some groups of clients who live near. Nevertheless, the overall cost of the routes is the lowest with the expression generated by the GP approach, which also utilises fewer vehicles for the operation.

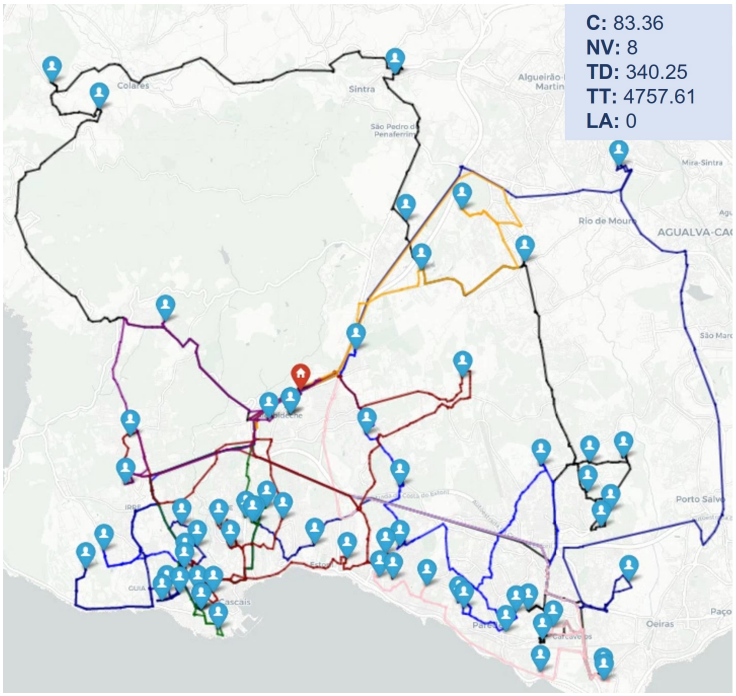


Figure 5.2: Example of a real solution generated with GP-RPB. The pins with a user icon symbolise the clients, while the pin with the house icon symbolises the depot. Each line colour represents a distinct route.

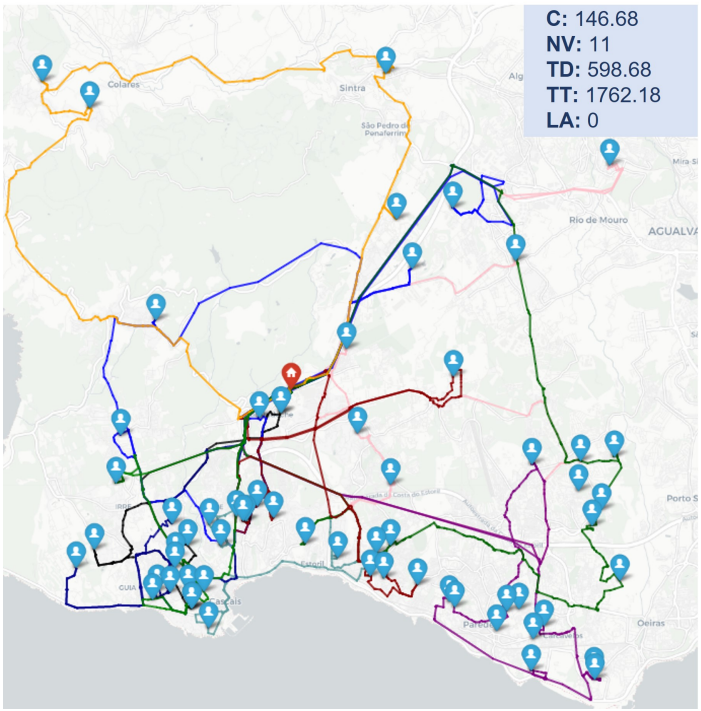


Figure 5.3: Example of a real solution generated by the retailer commercial software. The pins with a user icon symbolise the clients, while the pin with the house icon symbolises the depot. Each line colour represents a distinct route.

Chapter 6

Conclusions

This dissertation focused on the employment of a GP learning algorithm to generate routing policies which can be used to solve dynamic VRPs instances. A DVRPTW simulation model was developed in which, as new dynamic customer requests arrive, a solution to the problem is computed to try to include the new requests in the previously generated solution. The construction of the routes is made based on the routing policy evolved by the GP algorithm. The main purpose was to find policies that minimise the overall cost of the routes while being inherently interpretable.

6.1 Research Questions Remarks

Specifically, this dissertation explored the answer to the following research questions:

- **RQ1** - What are the most important features to consider when developing a real-world routing problem solution?

  Regarding this research question, at first, a deep analysis of the problem, its main characteristics and inherent restrictions was made to extract the most relevant features to be included in the evolutionary process. Next, the literature related to this topic, including the application of GP to other OR problems, was studied in order to try to find some features that could be adapted to our problem. This deconstruction resulted in three iterations of the guided empirical learning method proposed, in which three GP algorithm variants differing in the set of terminals were tested one at a time. The aim was, at each iteration, to guide the evolutionary process to a more promising search space region. In the end, the GP variant with the best set of terminals was submitted to a trim-down process to reduce the size of the set, eliminating the least performing variables. As a result of the investigation performed, the most important problem-related features to consider when developing a real-world routing problem solution comprise the following set: D, T, RT, AVGT, AVGTL, C, CR, RTT, NRTT.

  It is important to note that, the process of discovering the most relevant problem features to include in the policy learning algorithm is a very complex procedure, which can be explored

more deeply as future work. Considering that this step dictates which variables are part of the routing policy structure, they directly affect the quality of the solution found.

- **RQ2** - Can genetic programming approaches find (near-)optimal solutions for the dynamic vehicle routing problem with time windows?

  In what relates to this research question, it is difficult to give a clear conclusion. The literature instances utilised have been employed in studies that use other approaches to solve the VRP. However, these approaches are usually meta-heuristics, implying that a solution to a problem is exhaustively improved to the specific problem instances provided, and a lot of computation time is required for this operation. Accordingly, the results of these approaches are not comparable to the results of our work, where GP approaches generate routing policies, which are applied to never seen instances of the problem, giving almost instantaneously results. Still, although a fair comparison cannot be made between these approaches and our work, since better results than the ones achieved with our GP approach are available in the literature, it cannot be concluded that (near)-optimal solutions are being achieved when considering literature instances.

  On the other hand, when analysing the results achieved with real instances from the case study, the routing policy evolved with GP provides better results than the results of well-known manually designed heuristics. Consequently, it gives hope to the hypothesis of this question and perhaps near-optimal solutions, or at least good solutions, to the problem are being achieved.

  Concerning the quality of the solutions achieved, some improvements can be stated as future work, which may significantly impact the quality of the solutions. With more time available in the future, more tests could be executed with the GP-GOMEA algorithm with the GOMEA feature turned on and a higher number of individuals in the population. These tests will undoubtedly improve the performance of the routing policies found.

- **RQ3** - What is the trade-off between explainability and performance when employing GP-based approaches?

  Defining the level of explainability of an expression is a somewhat subjective concept. Generally, an expression can be defined as more easily comprehensible than others if its size is smaller than others. At the same time, an expression is said to be interpretable if a specialist can understand its nature simply by looking at it and comparing it to others.

  By analysing the expressions generated by the GP approaches, it is clear that higher performing policies also imply a higher number of nodes, reducing their explainability level. Concurrently, in all tests performed, the policies' sizes achieved were not drastically large, i.e. policy size was always lower than 63 nodes. This is due to the fact that the input parameters given to the GP algorithms did not allow for a large increase in the size of the expressions, thus preserving their comprehensibility.

Nevertheless, if we analyse the parcels that compose the routing policies, there are sometimes nonsense instances. The GP terminals used do not have all the same units, most of them have time units (T, RT, AVGT, AVGTL, RTT and NRTT), but others have distance units (D), cost units (C) or are adimensional (CR). The combination of variables with different dimensions can be hard to interpret, especially regarding addition or subtraction. Accordingly, in future work, employing Strongly-Typed GP (STGP) or Dimensionally Aware GP (DAGP) approaches can help deal with this issue, assuring consistency and improving the interpretability of policies evolved.

6.2 Final Remarks

Applying GP for dynamic real-life complex transportation problems is a rather young field of research. Consequently, the development of this work brings a significant contribution to the investigation field by achieving solutions' interpretability and real-time responses to VRP instances. Along with the answers to the research questions, other conclusions were drawn with the development of this work.

Regarding the most adequated GP algorithm to produce solutions for this transportation problem, tests were performed with a simpler version of the problem. The ECJ-GP, Tensor-GP and GP-GOMEA algorithms were put side-by-side in the same circumstances. The results achieved on a simulation environment of the TSPTW showed that GP-GOMEA, in both its versions: GP-Tree and GP-GOMEA, is the algorithm that finds the best routing policies. At the same time, the innovative GOMEA feature demonstrated a superior ability to explore the search space of problem solutions, having inherently a strength to combat the bloat limitation typical in GP. Despite the many advantages this GP algorithm brings, with its employment in our work, some limitations become apparent. Not all features offered in the algorithm could be utilised since some of them (e.g. Interleaved Multistart Scheme) required lots of resources, namely memory usage and computation time. Thus, the power of this algorithm could not be used at its maximum, bringing directions to future work. With more time and memory available, some tests should be carried out to check if better policies can be achieved by deeper exploring the GP-GOMEA's strengths. On the other hand, a characteristic not offered by this algorithm but that could improve its power is the possibility of entering an expression or a set of expressions as input and just performing its evaluation in a set of instances. Evidently, this is a feature we could add to the algorithm. However, given the complexity of this parse operation and the limited time available for the realisation of this dissertation, it was not implemented.

The obtained routing policies were tested with real-world data from a large European online retailer and proven to be a highly compensatory alternative for their route construction process. An improvement of 49.2% in the overall costs of the routes, with a reduction of 43.7% in the number of used vehicles, was verified when using a routing policy evolved by GP. Besides, a dynamic problem was solved. Thus, a replanning of the routes can be executed in real-time as a

new customer request arrives. Also, concerning the necessity of the case study in an efficient cost-to-serve model for the dynamic time slot model, the developed work is capable of filling this gap. The results of our approach were compared with the results of well-known heuristics typically applied to real-time transportation problems. It was shown that routing policies evolved by GP could successfully outperform the heuristics' results.

The developed model is transparent, reliable and flexible, being easily adapted to new problem characteristics or restrictions. Accordingly, for future work, it would be interesting to extend the work developed in this dissertation to other VRP variants, such as VRPDP or HFVRP.

References

[1] Vaishak Belle and Ioannis Papantonis. Principles and practice of explainable machine learning. *Frontiers in Big Data*, 4, 2021.

[2] Giulia Vilone and Luca Longo. Explainable artificial intelligence: a systematic review. *ArXiv*, 05 2020.

[3] Leonardo Augusto Ferreira, Frederico Gadelha Guimarães, and Rodrigo Silva. Applying genetic programming to improve interpretability in machine learning models. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, page 1–8. IEEE Press, 2020.

[4] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery.

[5] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc.

[6] Cristiane Ferreira, Gonçalo Figueira, and Pedro Amorim. Effective and interpretable dispatching rules for dynamic job shops via guided empirical learning. *Omega*, 111, 2022.

[7] Rui L. Lopes, Gonçalo Figueira, Pedro Amorim, and Bernardo Almada-Lobo. Cooperative coevolution of expressions for (r,q) inventory management policies using genetic programming. *International Journal of Production Research*, 58(2):509–525, 2020.

[8] Peer Kleinau and Ulrich Thonemann. Deriving inventory-control policies with genetic programming. *OR Spectrum*, 26:521–546, 10 2004.

[9] JM Link, PM Yager, JC Anjos, I Bediaga, C Castromonte, C Göbel, AA Machado, J Magnin, A Massafferri, JM De Miranda, et al. Application of genetic programming to high energy physics event selection. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 551(2-3):504–527, 2005.

[10] John R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, 1994.

[11] John R. Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3):251–284, 2010.

[12] Arne Strauss, Nalan Gülpınar, and Yijun Zheng. Dynamic pricing of flexible time slots for attended home delivery. *European Journal of Operational Research*, 294(3):1022–1041, 2021.

[13] Baris Yildiz and Martin Savelsbergh. Provably high-quality solutions for the meal delivery routing problem. *Transportation Science*, 53(5):1372–1388, 2019.

[14] Marius M. Solomon. On the worst-case performance of some heuristics for the vehicle routing and scheduling problem with time window constraints. *Networks*, 16(2):161–174, 1986.

[15] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

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

[17] 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.

[18] Shifeng Chen, Rong Chen, Gai-Ge Wang, Jian Gao, and Arun Sangaiah. An adaptive large neighborhood search heuristic for dynamic vehicle routing problems. *Computers & Electrical Engineering*, 67:596–607, 2018.

[19] R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati. Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization*, 10(4):327–343, 2005.

[20] Benjamin P. Evans, Bing Xue, and Mengjie Zhang. What's inside the black-box? a genetic programming method for interpreting complex machine learning models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1012—1020, New York, USA, 2019. Association for Computing Machinery.

[21] Ulf Johansson, Rikard König, and Lars Niklasson. The truth is in there - rule extraction from opaque models using genetic programming. In *FLAIRS Conference*, 01 2004.

[22] Shi-Yi Tan and Wei-Chang Yeh. The vehicle routing problem: State-of-the-art classification and review. *Applied Sciences*, 11(21), 2021.

[23] Josiah Jacobsen-Grocott, Yi Mei, Gang Chen, and Mengjie Zhang. Evolving heuristics for dynamic vehicle routing with time windows using genetic programming. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1948–1955, 2017.

[24] Joao Guilherme Cavalcanti Costa, Yi Mei, and Mengjie Zhang. Learning initialisation heuristic for large scale vehicle routing problem with genetic programming. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1864–1871, 2021.

[25] Shaolin Wang, Yi Mei, Mengjie Zhang, and Xin Yao. Genetic programming with niching for uncertain capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation*, 26(1):73–87, 2022.

[26] John Drake, Matthew Hyde, Khaled Ibrahim, and Ender Özcan. A genetic programming hyper-heuristic for the multidimensional knapsack problem. *Kybernetes*, 43:1500–1511, 11 2014.

[27] Leonardo Vanneschi and Riccardo Poli. Genetic programming - introduction, applications, theory and open issues. In *Handbook of Natural Computing*, pages 709–739, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[28] A.E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 08 1999.

[29] João Pedro Pedroso. Control of search parameters in evolutionary algorithms. In *Proceedings of the International Symposium on Nonlinear Theory and its Applications*, pages 1265–1268, 1997.

[30] Sean Luke and Lee Spector. A comparison of crossover and mutation in genetic programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 240–248. Morgan Kaufmann, 1997.

[31] David White and Simon Poulding. A rigorous evaluation of crossover and mutation in genetic programming. In *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, pages 220–231, 04 2009.

[32] Maarten Keijzer and Vladan Babovic. Dimensionally aware genetic programming. *Gecco-99: Proceedings of the Genetic and Evolutionary Computation Conference*, 01 1999.

[33] David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[34] Nicholas F McPhee Riccardo Poli, William B Langdon and John R Koza. *A field guide to genetic programming*. Lulu Enterprises, UK Ltd, 01 2008.

[35] Alain Ratle and Michèle Sebag. Genetic programming and domain knowledge: Beyond the limitations of grammar-guided machine discovery. In *Parallel Problem Solving from Nature PPSN VI*, pages 211–220, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[36] Jorge Tavares and Francisco B. Pereira. Designing pheromone update strategies with strongly typed genetic programming. In *Proceedings of the 14th European Conference on Genetic Programming*, EuroGP'11, page 85–96, Berlin, Heidelberg, 2011. Springer-Verlag.

[37] Phillip Wong and Mengjie Zhang. Algebraic simplification of gp programs during evolution. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, page 927–934, New York, NY, USA, 2006. Association for Computing Machinery.

[38] David Jackson. Promoting phenotypic diversity in genetic programming. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part II*, PPSN'10, page 472–481. Springer Berlin Heidelberg, 2010.

[39] Francisco Fernández de Vega, Gustavo Olague, Daniel Lanza, Francisco Chávez de la O, Wolfgang Banzhaf, Erik Goodman, Jose Menendez-Clavijo, and Axel Martinez. Time and individual duration in genetic programming. *IEEE Access*, 8:38692–38713, 2020.

[40] Leonardo Vanneschi, Mauro Castelli, and Sara Silva. Measuring bloat, overfitting and functional complexity in genetic programming. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, page 877–884, New York, NY, USA, 2010. Association for Computing Machinery.

[41] Torsten Hildebrandt and Juergen Branke. On using surrogates with genetic programming. *Evolutionary computation*, 23, 06 2014.

[42] Sean Luke. Ecj then and now. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, page 1223–1230, New York, NY, USA, 2017. Association for Computing Machinery.

[43] Francisco Baeta, João Correia, Tiago Martins, and Penousal Machado. Speed benchmarking of genetic programming frameworks. In *GECCO 2021 – Proceedings of the 2021 Genetic and Evolutionary Computation Conference*. ACM, 2021.

[44] Francisco Baeta, João Correia, Tiago Martins, and Penousal Machado. Tensorgp – genetic programming engine in tensorflow. In *Applications of Evolutionary Computation: 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, Virtual Event, April 7–9, 2021, Proceedings*, page 763–778, Berlin, Heidelberg, 2021. Springer-Verlag.

[45] William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. Contemporary symbolic regression methods and their relative performance. *ArXiv*, 2021.

[46] Marco Virgolin, Tanja Alderliesten, Cees Witteveen, and Peter A. N. Bosman. Scalable genetic programming by gene-pool optimal mixing and input-space entropy-based building-block learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, page 1041–1048, New York, NY, USA, 2017. Association for Computing Machinery.

[47] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[48] Merrill M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956.

[49] 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.

[50] Nicolas Jozefowiez, Frédéric Semet, and El-Ghazali Talbi. Multi-objective vehicle routing problems. *European Journal of Operational Research*, 189(2):293–309, 2008.

[51] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science*, 39:104–118, 02 2005.

[52] Binbin Pan, Zhenzhen Zhang, and Andrew Lim. Multi-trip time-dependent vehicle routing problem with time windows. *European Journal of Operational Research*, 291(1):218–231, 2021.

[53] Serap Ercan Cömert, Harun Yazgan, İrem Sertvuran, and Hanife Sengul. A new approach for solution of vehicle routing problem with hard time window: an application in a supermarket chain. *Sādhanā*, 42(12):2067–2080, 2017.

[54] Florian Arnold and Kenneth Sörensen. What makes a vrp solution good? the generation of problem-specific knowledge for heuristics. *Computers & Operations Research*, 106:280–288, 2019.

[55] Harilaos N. Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980.

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

[57] Lianxi Hong. An improved lns algorithm for real-time vehicle routing problem with time windows. *Computers & Operations Research*, 39(2):151–163, 2012.

[58] Francesco Ferrucci, Stefan Bock, and Michel Gendreau. A pro-active real-time control approach for dynamic vehicle routing problems dealing with the delivery of urgent goods. *European Journal of Operational Research*, 225(1):130–141, 2013.

[59] Jesica Armas and Belén Melián-Batista. Constrained dynamic vehicle routing problems with time windows. *Soft Computing*, 19, 01 2015.

[60] Soumia Ichoua, Michel Gendreau, and Jean-Yves Potvin. Exploiting knowledge about future demands for real-time vehicle dispatching. *Transportation Science*, 40:211–225, 05 2006.

[61] K. Lund, O.B.G. Madsen, J.M. Rygaard, Danmarks Tekniske Universitet. Institut for Matematisk Modellering, and Technical University of Denmark. Institute of Mathematical Modelling. *Vehicle Routing Problems with Varying Degrees of Dynamism*. IMM-REP. IMM, Institute of Mathematical Modelling, Technical University of Denmark, 1996.

[62] Allan Larsen. *The dynamic vehicle routing problem*. PhD thesis, Institute of Mathematical Modelling, Technical University of Denmark, 01 2000.

[63] Soumia Ichoua, Michel Gendreau, and Jean-Yves Potvin. *Planned Route Optimization For Real-Time Vehicle Routing*, volume 38, pages 1–18. Springer US, Boston, MA, 05 2007.

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

[65] Marcin Woch and Piotr Łebkowski. Sequential simulated annealing for the vehicle routing problem with time windows. *Decision Making in Manufacturing and Services*, 3(2):87–100, Dec. 2009.

[66] Hamida Abidi, Khaled Hassine, and Fethi Mguis. Genetic algorithm for solving a dynamic vehicle routing problem with time windows. In *2018 International Conference on High Performance Computing Simulation (HPCS)*, pages 782–788, 2018.

[67] Jianhua Xiao, Tao Zhang, Jingguo Du, and Xingyi Zhang. An evolutionary multiobjective route grouping-based heuristic algorithm for large-scale capacitated vehicle routing problems. *IEEE Transactions on Cybernetics*, 51(8):4173–4186, 10 2019.

[68] Florentin D. Hildebrandt, Barrett W. Thomas, and Marlin W. Ulmer. Where the action is: Let's make reinforcement learning for stochastic dynamic vehicle routing problems work! *arXiv preprint arXiv:2103.00507*, 2021.

[69] Mohammadreza Nazari, Afshin Oroojlooy, Martin Takáč, and Lawrence V. Snyder. Reinforcement learning for solving the vehicle routing problem. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, page 9861–9871, NY, USA, 2018. Curran Associates Inc.

[70] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer Berlin Heidelberg, 01 2004.

[71] Ángel Corberán, Richard Eglese, Geir Hasle, Isaac Plana, and José María Sanchis. Arc routing problems: A review of the past, present, and future. *Networks*, 77(1):88–115, 2021.

[72] Gianpaolo Ghiani, Emanuele Manni, Antonella Quaranta, and Chefi Triki. Anticipatory algorithms for same-day courier dispatching. *Transportation Research Part E Logistics and Transportation Review*, 45:96–106, 01 2009.

[73] Verena Schmid. Solving the dynamic ambulance relocation and dispatching problem using approximate dynamic programming. *European Journal of Operational Research*, 219:611–621, 06 2012.

[74] Martin Desrochers and Gilbert Laporte. Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. *Operations Research Letters*, 10(1):27–36, 1991.

[75] Raluca Necula, Mihaela Breaban, and Madalina Raschip. Tackling dynamic vehicle routing problem with time windows by means of ant colony system. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2480–2487, 2017.

[76] Andreas Lackner. *Dynamische Tourenplanung mit ausgewählten Metaheuristiken*. PhD thesis, GeorgAugust-Universität Göttingen, 2004.

[77] M. Virgolin, T. Alderliesten, C. Witteveen, and P. A. N. Bosman. Improving model-based genetic programming for symbolic regression of small expressions. *Evolutionary Computation*, 29(2):211–237, 2021.

[78] Lan Peng and Chase Murray. Veroviz: A vehicle routing visualization toolkit. *SSRN Electronic Journal*, 01 2020.