Hyper-Heuristics for Automatic Configuration of Local Search-based Heuristics for the Large-Scale Vehicle Routing Problem

by

João Guilherme Cavalcanti Costa

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2023

Abstract

In this thesis we tackle one of the issues of a well-known hard-to-solve problem. The problem is the Vehicle Routing Problem (VRP), specifically in its large-scale version, also known as the Large Scale VRP (LSVRP). The LSVRP has a number of customers much larger than what traditional approaches can solve with ease, bringing extra challenges due to the amount of resources required to compute the solutions. Even when considering non-exact approaches, the scale of the problem demands reductions to the size of the solution search space. A challenge arises when doing this reduction while trying to maintain most or all of the very good solutions.

The issue we tackle is the number of manual decisions that exist in current method design. Most of the existing literature for the VRP utilises decisions that are seemingly arbitrary or based on expertise design, sometimes with little experimentation. This leads to methods that fail to generalise well for some instances or, especially, for large-scale problems. Considering the goal of minimisation problems, such as the VRP, to be reducing costs as much as possible, in several scenarios these manually designed fixed decisions often lead to a higher cost. We argue that it is possible to improve the effectiveness of the methods without manually setting up parameter values for each instance.

We approach this issue by considering the local search framework, which is arguably the most used engine in most methods for the VRP. We then consider its main design components: initialisation, improvement and acceptance. We apply several machine learning and evolutionary computation approaches as hyper-heuristics to reduce the amount of manual decisions in these three design components.

Hence, the overall goal of this thesis is to build hyper-heuristics as techniques for automatically improving and configuring local search-based methods in the context of the LSVRP.

For the initialisation step, first we consider its impact to the search process by analysing several baseline methods' performances. We then introduce ways to utilise known and new features to predict best performing existing heuristic or build new solutions that can be easily improved. For predicting, we utilise several machine learning techniques to validate the results independently. The results show that cost is not the main feature in an initial solution. We show that some other characteristics, such as the compactness or width of a solution, have a stronger correlation than cost, leading to faster or better improvement phases.

This thesis also develops three evolutionary hyper-heuristic methods to automate the improvement phase. We consider new chromosomes incorporating pruning strategies that evolve to automatically design a heuristic configuration. These strategies minimise the amount of manual decisions leading to more generalisable methods. The results show that the improvement phase can be positively affected when automatically optimised, often outranking the manually designed methodologies.

We also consider an adaptive heuristic strategy which improves the efficiency of the local search. We apply this stochastic online approach to a robust framework, increasing its effectiveness due to the extra efficiency. Among the results, we observe a significant increase in the number of iterations given the same time-frame. We also observe a significant cost reduction for most instances considered, especially very-large cases. The proposed strategy also works independently of the instance, increasing the framework's generality.

The fourth and final contribution regards how to predict which acceptance criteria can be used to escape local optima more effectively. We utilise machine learning and evolutionary computation to make such predictions by considering the same robust local search-based framework.

The problem was modelled as both classification and regression tasks. The latter was added in an attempt to avoid bias on the labelled data. However, the results show that this task is difficult to correlate and predict. We are still able to find success for several cases, improving the quality in a number of scenarios.

In summary, this thesis develops strategies and methods that can be used in combination with existing and new local search-based methods to solve the LSVRP. The developed techniques have shown ability to reduce the search space effectively and improve the efficiency of the considered approaches for several cases, whilst minimising manual decisions in method design.

To my dear friend Apsara.

Acknowledgments

First, I would like to acknowledge my Mother and Grandmother, Ana Cristina and Maria da Paz. They were the biggest believers and investors of my career and without them I would not be here. Thank you for being so supportive. I love both of you.

My uppermost appreciation go to my supervisors, A/Prof. Yi Mei and Prof. Mengjie Zhang, whose opinions, discussions, contributions and guidance all led to this thesis, thank you. I also would like to acknowledge the contributions from our research groups, ECRG and ECCO, with valuable feedback during presentation practices and discussions. The faculty from the university also played a big role in my studies, with the support from Victoria University of Wellington for providing the necessary scholarships and financial support, as well as their student support teams from career, learning and health.

My family, especially my father José Enéas, my brother Enéas Filho and my sister Ana Luíza (and our dog Lola). My father family-side, the Costa family and my mother family-side, the Caldeiras, especially my aunt Marlene and my uncle Alex. Thank you for your support and love, especially for enduring my absence for long periods of time.

From my Masters degree in USP, with support both academically and by writing superb recommendation letters that allowed me to reach this place, my then supervisor Prof. Maristela Oliveira dos Santos, with a special mention to Prof. Cláudio Toledo. Thank you.

To the people that kept me sane through the little matches, Bruno "Bat-

man" Vicente, Francisco "Bill" Rocha, Diego "Dispew" Porto, Thiago "Dexther" Santos and Ranulfo "Rufinho" Plutarco, with a bonus for some high-level discussions about my topic which helped in certain times, which never left me wondering whether can we do it. These guys are good.

To my friends in Brazil and to those who helped me make my moving to New Zealand smoother, especially Yamila Larisse, Roney Lira, João Paulo, Ana Caroline, Matheus, Helder Eiki, Brenda Marye, as well as all the other guys from RepK2 and friends from Sanca city, Hanah, Mateus, class of 2011.1 and other friends in Teresina, I miss all of you more than half of what you deserve.

In this amazing place called New Zealand, I also found a family in Baligh, Harisu, Hiroshika (the three buffers), Isabella and Xiaohan. You guys are the reason I did not go crazy doing this thing. I love all of you. And to Prajakta Ujagare, who endured my ups and downs more than anyone.

Here in Wellington my friends and flatmates which helped me make this PhD student life a bit easier: Fangfang, Mazhar, Mahdi, Abbasi, Kirita, Victoria, Muru, Duncan, Nora, Zhixing, Junhao, Qinglan, Jordan, Buddhima, Agatha, Atefeh, Max, Ying, Kosi, Ramya, Duleeka, Junaid, Maryam, Hengzhe, Maeva, Yulia, Kama and so many others that were there with me in this quest.

Finally, I want to acknowledge the person that made me start to care about my academic life again. Your passion for this life made a light burn inside of me again. I wish you were still here, as this was your dream even more than mine. Thank you, Apsara Wimalasiri. May you rest in peace.

Publications

1. **Joao Guilherme Cavalcanti Costa**, Yi Mei, Mengjie Zhang. Cluster-based Hyper-Heuristic for Large-Scale Vehicle Routing Problem. In 2020 IEEE Congress on Evolutionary Computation (CEC 2020) (pp. 1-8). IEEE

2. **Joao Guilherme Cavalcanti Costa**, Yi Mei, Mengjie Zhang. Adaptive Search Space through Evolutionary Hyper-Heuristics for the Large-Scale Vehicle Routing Problem. IEEE Symposium Series on Computational Intelligence (SSCI 2020) (pp. 2415-2422). IEEE

3. **Joao Guilherme Cavalcanti Costa**, Yi Mei, Mengjie Zhang. An Evolutionary Hyper-Heuristic Approach to the Large Scale Vehicle Routing Problem. IEEE Congress on Evolutionary Computation (CEC 2021) (pp. 2109-2116). IEEE

4. **Joao Guilherme Cavalcanti Costa**, Yi Mei, Mengjie Zhang. Learning Initialisation Heuristic for Large Scale Vehicle Routing Problem with Genetic Programming. IEEE Congress on Evolutionary Computation (CEC 2021) (pp. 1864-1871). IEEE

5. **Joao Guilherme Cavalcanti Costa**, Yi Mei, Mengjie Zhang. Learning Penalisation Criterion of Guided Local Search for Large Scale Vehicle Routing Problem. IEEE Symposium Series on Computational Intelligence (SSCI 2021) (pp. 1-8). IEEE

6. **Joao Guilherme Cavalcanti Costa**, Yi Mei, Mengjie Zhang. Guided local search with an adaptive neighbourhood size heuristic for large scale vehicle routing problems. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2022) (pp. 213-221). ACM

7. **Joao Guilherme Cavalcanti Costa**, Yi Mei, Mengjie Zhang. Learning to Select Initialisation Heuristic for Vehicle Routing Problem. (Accepted by GECCO 2023) 9 pp.

Contents

List of Algorithms **xvi**

List of Figures **xix**

List of Tables **xxiii**

1 Introduction **1**

1.1 Problem Statement 2

1.1.1 Manually Designed Decisions in Local Search 5

1.1.2 Dealing with Large Search Space in Local Search . . . 8

1.2 Motivations 9

1.2.1 Complexity of the Heuristic Search Space 9

1.2.2 Initialisation 13

1.2.3 Improvement — Operator Selection Strategies 14

1.2.4 Improvement — Operator Search Space Size 16

1.2.5 Evaluation and Acceptance Criteria 17

1.3 Research Goals 19

1.4 Major Contributions 22

1.5 Organisation of Thesis 25

2 Literature Review **29**

2.1 Basic Concepts 29

2.1.1 Combinatorial Optimisation Problem 29

2.1.2 Neighbourhood 30

2.1.3 Local and Global Optima 31
2.1.4 Local Search 31
2.1.5 Exploration vs Exploitation 33
2.1.6 Exact methods vs Heuristic vs MetaHeuristic vs Hyper-Heuristic 34
2.2 Vehicle Routing Problem 36
2.2.1 Large-Scale Vehicle Routing Problem 39
2.3 Hyper-Heuristics 41
2.3.1 Classification of Hyper-Heuristics 42
2.3.2 Machine Learning as HH 43
2.4 Evolutionary Algorithms 44
2.4.1 Evolutionary Hyper-Heuristics 46
2.5 Related Work 48
2.5.1 Exact Methods for the VRP 48
2.5.2 Heuristics for the VRP 50
2.5.3 Metaheuristics for the VRP 54
2.5.4 Addressing the LSVRP Heuristically 58
2.5.5 Hyper-Heuristics for the VRP and LSVRP 63
2.6 Summary 66

3 Learning Effective Initialisation 69
3.1 Introduction 69
3.1.1 Chapter Goals 70
3.1.2 Chapter Organisation 71
3.2 The Initial Solution Impact 71
3.2.1 The KGLS and the Initialisation Heuristics 71
3.2.2 Analysing the Initialisation Heuristics Performance . 73
3.3 Learning to Select Initialisation Method 79
3.3.1 Labelling the Data 79
3.3.2 Machine Learning Approach 81
3.3.3 Experiment Design 82

3.3.4 Results and Discussions 83
3.3.5 Further Tests and Feature Analysis 86
3.3.6 Summary 88
3.4 GPHH for Effective Initialisation 89
3.4.1 Novel Terminals using the Width 89
3.4.2 Incorporating Width to the Fitness Function 90
3.4.3 Route Construction 92
3.4.4 Genetic Operators 93
3.4.5 Experiment Design 94
3.4.6 Results and Discussions 97
3.4.7 Summary 104
3.5 Chapter Summary 105
3.5.1 Consideration on Scalability and Computational time 106

4 Operator Selection and Bounds with Evolutionary Hyper-Heuristics 107
4.1 Introduction 107
4.1.1 Chapter Goals 109
4.1.2 Chapter Organisation 110
4.2 Cluster-based Hyper-Heuristics Approach 110
4.2.1 Solution Representation 112
4.2.2 CbHH Framework 114
4.2.3 Genetic Component 117
4.2.4 Experiment Design 119
4.2.5 Experiment Goals 120
4.2.6 Dataset and Parameters 121
4.2.7 Results and Discussions 122
4.2.8 Summary 125
4.3 The Dual-Layered Chromosome GA 126
4.3.1 Solution Representation 126
4.3.2 Evolutionary Hyper-Heuristic 127
4.3.3 Experiment Design 131

4.3.4 Results and Discussions . . . 133
4.3.5 Effect of initial solution . . . 133
4.3.6 Effectiveness of the adaptive size limit . . . 134
4.3.7 Comparison with other methods . . . 136
4.3.8 Further Analysis of the evolved solutions . . . 136
4.3.9 Summary . . . 138
4.4 Evolutionary Hyper-Heuristic for Knowledge-Guided Local Search . . . 139
4.4.1 Representation . . . 139
4.4.2 The KGLS badness functions . . . 141
4.4.3 GA Hyper-Heuristics . . . 142
4.4.4 Experiment Design . . . 146
4.4.5 Training and Testing . . . 148
4.4.6 Parameters . . . 149
4.4.7 Results and Discussions . . . 150
4.4.8 Summary . . . 154
4.5 Chapter Summary . . . 155
4.5.1 Consideration on Scalability and Computational time 156

5 Learning to Guide Solution Search Space 159
5.1 Introduction . . . 159
5.1.1 Chapter Goals . . . 161
5.1.2 Chapter Organisation . . . 161
5.2 Proposed Method . . . 162
5.2.1 Overall Framework . . . 162
5.2.2 Neighbourhood Size Adaption Heuristic . . . 163
5.3 Experiment Design . . . 167
5.4 Results and Discussions . . . 168
5.5 Further Analysis . . . 174
5.5.1 Efficiency . . . 174
5.5.2 Performance Instance-wise . . . 176

5.5.3 Neighbourhood Pruning Accuracy 176
5.6 Chapter Summary 178
5.6.1 Consideration on Scalability and Computational time 180

6 Learning Strategies to Escape Local Optimum 181
6.1 Introduction 181
6.1.1 Chapter Goals 183
6.1.2 Chapter Organisation 183
6.2 Learning Instance-Specific Penalisation Criterion 184
6.2.1 Data Generation 187
6.2.2 Classification Model 188
6.2.3 Regression Model 188
6.2.4 Hybrid Regression-Classification Model for GP . . . 190
6.3 Experiments 190
6.4 Results and Discussions 192
6.4.1 Preliminary analysis on the original data 192
6.4.2 Results for different splits 193
6.4.3 Training Performance 194
6.4.4 Test Performance 195
6.5 Chapter Summary 198
6.5.1 Consideration on Scalability and Computational time 199

7 Conclusions 201
7.1 Research Questions and Main Conclusions 202
7.1.1 Learning Effective Initialisation 202
7.1.2 Operator Selection and Bounds with Evolutionary Hyper-Heuristics 204
7.1.3 Learning to Guide Solution Search Space 205
7.1.4 Learning Acceptance Strategies to Escape from Local Optima 207
7.2 Future Work 208
7.2.1 Learning Effective Initialisation 208

7.2.2 Operator Selection and Bounds with Evolutionary Hyper-Heuristics 209
7.2.3 Learning to Guide Solution Search Space 210
7.2.4 Learning Acceptance Strategies to Escape from Local Optima 211
7.2.5 New Directions 211

8 Bibliography **213**

List of Algorithms

1.1 The Local Search Framework for the VRP 5
2.1 Local Search Algorithm 32
2.2 Evolutionary Algorithm Overall Scheme 45
3.1 Route construction method. 93
3.2 Genetic Programming training method. 93
4.1 Clustering-based Selection Perturbative Hyper-Heuristic . . 116
4.2 Low-Level Heuristic Phase 116
4.3 Cluster Update Phase 117
4.4 Adaptive-based Selection Perturbative Hyper-Heuristic . . . 128
4.5 Low-Level Heuristic Phase 129
4.6 Hyper-Heuristic Genetic Algorithm's Training Phase. 144
4.7 KGLS* used to evaluate the individuals. 147
5.1 Newly Proposed Method 163
5.2 $S' \leftarrow \mathrm{op}(S, \Delta)$ 164
5.3 $\texttt{AdaptNS}(\Delta, S, S', \Omega)$ 165

List of Figures

1.1 Outline of thesis chapters and main contributions. 26

2.1 Example of how the VRP solution looks like. A set of vertexes are visited in some sequence. 37
2.2 Classification of Hyper-Heuristics according to [28]. 42
2.3 Examples of representation and basic operations in a Genetic Algorithm. 46
2.4 Representation of a Genetic Programming Tree. 47
2.5 Examples of Genetic Programming Crossover and Mutation operators. 47
2.6 A simple example of the 2-Opt* operation. 53

3.1 Example of routes created by the 4 heuristics considered — for instance X-n176-k26: (a) CW100 (with cost 52570) (b) CW (with cost 52551) (c) 1-NN (with cost 66134) (d) Sweep (with cost 95138) 74
3.2 An overall flowchart for the: (a) Training phase (b) Test phase 82
3.3 The overall performance of the ML methods over the 30 runs by two different statistics: (a) Accuracy (b) GAP in solution cost to optimal classification. 85
3.4 The overall accuracy and performance of the ML methods over the 30 runs considering the Li and Golden datasets. . . 87
3.5 Feature ranking from one of the runs of the Random Forest. 88

3.6 Examples of how the penalisation criteria guides the solution into new local optima. 91
3.7 Comparison between the proposed GPHH average and best case to the KGLS* performance. 101
3.8 Comparison between the GPHH^2 average and best case to the KGLS* performance. 103
3.9 Evolved tree with the new operators. 104

4.1 A fictional example of how the cluster-based search adaptively changes the cluster space. 112
4.2 Example of a CbHH chromosome. 112
4.3 Overall flow chart of the proposed framework. 114
4.4 Example of a best-best crossover, based on [85]. 119
4.5 An example of chromosome for the Genetic Algorithm. . . . 127
4.6 Example of a two-point crossover for the two-dimensional chromosome. 129
4.7 Example of the mutation operators. On the left, the worst improving sequence is removed. On the right, new random LLHs are added. 130
4.8 Example of an evolved chromosome. 138
4.9 Chromosome for our evolutionary hyper-heuristic. 140
4.10 Flowchart for the HH framework. 143
4.11 Flowchart for the GLS framework. 145
4.12 Improvement over time for the KGLS* implementation considering the test instance set. 148
4.13 Example of heuristic evolved individual. 154
4.14 Performance of the final heuristic over time, for all 30 runs. . 155

5.1 The index of the best neighbour for the *Cross-Exchange* in 50 iterations of KGLS. 166
5.2 The index of the best neighbour for the *Relocation Chain* in 50 iterations of KGLS. 166

5.3 Solution progress curve of a single run of the KGLS and the 30 runs of the proposed algorithm (PH) on a *very-large* instance. 173
5.4 Convergence curve of a single run of the KGLS and the 30 runs of the proposed algorithm (PH) on a *large* instance. . . . 174
5.5 Convergence curve of a single run of the KGLS and the 30 runs of the proposed algorithm (PH) on a *medium* instance. . 175
5.6 Index of the best neighbour for the *CE* operator in each iteration by KGLS and the proposed algorithm (PH). 178
5.7 Index of the best neighbour for the *RC* operator in each iteration by KGLS and the proposed algorithm (PH). 179

6.1 Training model used for the Machine Learning approaches. . 189
6.2 An example of training instance considered. The red exes represent the actual improvement values, while the blue dots represent the GP regression prediction. 197
6.3 Box plot for the performance of the different ML method used with outliers removed. The values indicate the distance to the BKS, therefore, the lower the better. 198

List of Tables

3.1 The initial costs given by the different initialisation heuristics for some representative instances. The average represents the value over the 100 instances of [183]. 75

3.2 The results obtained by KGLS with different initial solutions on some representative instances. The average represents the value over the 100 instances of [183]. 77

3.3 A summary of additional statistics for the preliminary experiments over the 100 instances of [183]. 77

3.4 Solution-specific ($*$) and Instance-specific ($+$) features. All features are proposed in [11], except for Cost and NTS. . . . 80

3.5 Results for all the considered ML methods and their respective costs for one of the runs with 30 randomly selected test instances. They are also compared to the default mode (using only CW100) and to a **theoretical** 100% accuracy method. 84

3.6 Number of instances that each initialisation heuristic outperforms the others, Li [118] and Golden [80] datasets. . . . 86

3.7 Set of terminals for the GPHH 94

3.8 Knowledge-Guided Local Search parameters. 95

3.9 Genetic Programming parameters. 96

3.10 Results compared to the Best Known Solution (BKS). In bold show significance in the comparison between the KGLS* and our GPHH. Underlined are the overall best. 97

3.10 Results compared to the Best Known Solution (BKS). In bold show significance in the comparison between the KGLS* and our GPHH. Underlined are the overall best (cont.). . . . 98
3.10 Results compared to the Best Known Solution (BKS). In bold show significance in the comparison between the KGLS* and our GPHH. Underlined are the overall best (cont.). . . . 99
3.10 Results compared to the Best Known Solution (BKS). In bold show significance in the comparison between the KGLS* and our GPHH. Underlined are the overall best (cont.). . . . 100
3.11 Results compared to the Best Known Solution (BKS) for the two compared GPHHs. 101
3.11 Results compared to the Best Known Solution (BKS) for the two compared GPHHs (cont.). 102
3.11 Results compared to the Best Known Solution (BKS) for the two compared GPHHs (cont.). 103

4.1 Default parameters. 121
4.2 Results for Experiment 1: the effectiveness of the proposed method when changing the search space size. The average solution, as well as the standard deviation and best solution, are shown, followed by execution time in minutes. . . . 122
4.3 Results for Experiment 2: a comparison between both methods of initialization. The solution average and standard deviation are shown. 123
4.4 Results for Experiment 3: a comparison between the two types of decoding for the chromosome. Showing the average, standard deviation and execution time in minutes. . . . 124
4.5 Results for Experiment 4: comparison between different BKS and different algorithms with the proposed method. The best, the average with standard deviation, as well as the GAP from the best solution to the BKS are shown. BKS with (*) indicate that they are the optimal. 125

4.6 Wicolxon Rank-Sum test: p-values of the comparison between the base method (CbHH 25%) and the other evaluated variations. 125
4.7 Genetic Algorithm parameters. 133
4.8 Results comparing the initial solution impact. Execution time in minutes 134
4.9 Results comparing a fixed search space size and an adaptive one. Execution time given is in minutes 135
4.10 A comparison between BKS and the manually designed heuristics with the proposed method. The GAP shows how close to the BKS the average solution is. BKS with (*) indicates that they are the optimal. The best solution for each instance is highlighted. The Savings algorithm from [37] (CW) is shown as a point of comparison. 137
4.11 The average neighbourhood limit for each Low-Level Heuristic and the average in total for each given instance. 138
4.12 The GA Hyper-Heuristics parameters. 150
4.13 Results compared to BKS. In bold are the best between KGLS* and EHH, while underlined are the overall best. 151
4.13 Results compared to BKS. In bold are the best between KGLS* and EHH, while underlined are the overall best (cont.). . . . 152
4.13 Results compared to BKS. In bold are the best between KGLS* and EHH, while underlined are the overall best (cont.). . . . 153

5.1 Parameter settings 168
5.2 Results of 30 runs on the *Medium* instances. 168
5.2 Results of 30 runs on the *Medium* instances (cont.). 169
5.2 Results of 30 runs on the *Medium* instances (cont.). 170
5.3 Results of 30 runs on the *large* instances. 170
5.3 Results of 30 runs on the *large* instances (cont.). 171
5.3 Results of 30 runs on the *large* instances (cont.). 172
5.4 Results of 30 runs on the *very-large* instances. 172

5.5 Comparison on the average number of iterations for the KGLS and the proposed heuristic. 175

5.6 Breakdown of instance characteristics for the solutions that got worse. 177

5.7 Accuracy of the operators CE and RC versus the default pruning size. 177

6.1 Some examples of the difference in performance between the different penalisation criterion in the KGLS that runs for 10 minutes. Average and Total presented are across all 100 instances from the dataset. 185

6.2 Parameters considered for the experiments. In bold are the default configuration. 192

6.3 Number of instances that each criteria has performed the best. Values represent their relative difference to the default criteria (K). Taken from a 10 minutes execution of the KGLS among all 100 instances. 193

6.4 Summary of the performance when comparing two different splits on the dataset. Values are **GAP** to the BKS, averaged for the 30 runs and the respective instances. Column Best is the best run out of the 30. 194

6.5 The training results. Presented values are the distance to the BKS, averaged out across the training data. Columns 1-4 shows the values for the KGLS criterion. Column Best shows the target value for the ML methods. Next columns represent the average and best over the 30 runs for the ML algorithms studied. 195

6.6 The results for a subset of the test set. Presented values are the distance to the BKS. Columns 2-5 shows the criteria of the KGLS run. Best shows the best values from each criterion. Next columns represent the average expected GAP over the 30 runs for the ML algorithms. Finally, last rows present the average and best results over the 79 test instances. 196
6.7 Number of instances the models have outperformed the default criteria K, out of the 79 test instances. Average and best case over 30 runs. 197

Chapter 1

Introduction

Recent increases in e-commerce and other related activities have led to an expansion of most supply-chain operations. One of the steps in this chain, and perhaps the most influential one, is **transportation**, which takes up to one-third of the logistics costs [203]. Roughly speaking, in the supply-chain the transportation is the process of moving goods or produce, from suppliers to consumers. It starts from transporting the raw materials from their respective sites of origin to the industries which processes them. Then, a final product is transported from those industries to stores or distribution centres, which can then be delivered to the final **customers**[1] (this step is also known as *last-mile* delivery). In several cases, there are additional steps before delivering to the final customers, for example, transporting the products from distribution centres to different warehouses, which are smaller and closer to said customers. No matter how many substeps, the overall idea of transportation is the same: moving items from one place to another, usually requiring the use of vehicles (such as trucks, trains, ships, car or even bikes), until they reach their final destination.

When looking at its direct negative impact to society, the transportation

[1]Customers in this process can be companies, other industries or individuals, and will be considered as the same throughout this thesis, as distinguishing the products' end goals is not in beyond the scope of this thesis.

phase affects the overall population the most. Although most of the transportation occurs within industries and other large companies, the use of typical large vehicles, such as trucks, airplanes and ships can overflow the general transportation grid. These vehicles directly or indirectly affect local traffic, air and water quality levels (due to most large vehicles also producing more pollution gases) — consequences that are felt by everyone. Not to mention the prices of goods and products are directly impacted by the effectiveness of the transportation phase. These points (and more) can be seen in the reviews [87, 106].

In this thesis, we consider one strategy that is viable to be implemented without drastic changes in the existing systems: **finding better routes**. Each cargo transported can usually be done by different paths. Since there are so many, there is a positive ripple effect if these paths are optimised to minimise their distances. The use of more efficient routes can significantly reduce costs and have a direct repercussion to the aforementioned impacts, whether being by reducing the operation costs or by minimising the overflow on the transportation grid — this problem is known in the scientific literature as the **Vehicle Routing Problem** (VRP).

One of the main challenges of the VRP regards a large number of customers, which becomes a computational burden due to the complex network. Another challenge regards the manually designed decisions in the development process, requiring a great deal of expertise to set-up a good method, which might not even work well for different scenarios. Hence, this thesis focuses on this optimisation task in large-scale scenarios and on how to find more automatic methods which can learn how to solve it.

1.1 Problem Statement

The formal definition for the VRP is based on the work of [46] in 1959, which has been extensively studied since then. The traditional VRP can be represented by a graph $G = (V, E)$, where the set of vertexes $V =$

$\{0, 1, 2, \cdots, n\}$ represents the customers, each with a demand q_i that needs to be fulfilled. There is a special point, usually the node 0, which is the depot, and whose demand $q_0 = 0$. The set of edges $E = \{(i, j) \in E, i \neq j\}$ expresses the selected paths between customers' pairs. In the classical VRP, all nodes are fully connected and the space is Euclidean[2]. A solution to the VRP needs to contain a set of routes which visit every customer, attending every demand without exceeding any vehicle's capacity Q. The overall objective is to find the routes which minimise the total distance travelled.

The VRP is a *combinatorial optimisation problem* (COP) belonging to the class of $\mathcal{NP}$**-hard** problems, as proven in [117]. The VRP and variants have an exponential number of possible solutions mainly based on the number of nodes. When the instance being solved has a large number of customers, the problem can be referred as the Large-Scale VRP (LSVRP). Today[3], that number is arguably considered to be more than 200 [9,75,95, 167]. This number could be related to how effective exact approaches are in solving problems of up to this size, meaning it might keep changing with the years. For this thesis we consider 200 to be the lower bound on what constitutes a large scale problem, with anything between 100 and 200 being considered an intersection zone between large and small, similar to what was done in [6].

Although claimed as *large*, real applications can easily reach up to thousands of customers. Such sizes make it very hard to find a balance between **exploration** and **exploitation**. These two concepts are related to how to search for solutions. Exploration is concerned with the finding of good areas of the search space, while exploitation aims to find the best solution in the nearby region. The most efficient algorithms will correctly harmonise these two orthogonal investigation strategies.

The methods which solve the VRP can be divided into two categories:

[2]This implies that the distance $d(a, b) + d(b, c) \leq d(a, c), \forall (a, b, c) \in V$

[3]Some authors claim that 100 customers is enough to classify an instance as Large-Scale, as in [106], however that seems less common in recent literature.

exact and non-exact [73, 111], although they are not exclusive, as it is possible to combine them. The exact methods, such as Branch-and-Price or Column Generation, will find the mathematically proven optimal solution for the problem, i.e. they find the best possible solution [73]. The exact methods will perform both exploration and exploitation to their fullest, covering every region of the search space through (usually implicit) enumeration. This task, however, will come at a very high cost of computational resources such as memory and, especially, time. Alternatively, the non-exact[4] methods aim to find good solutions within a reasonable time[5].

Typically, these non-exact approaches only search throughout some sub-regions of the solution search space, utilising some sort of **randomisation** or some **expert-designed** decision to define those regions. Here, heuristics and metaheuristics are the go-to-method, with the latter taking dominance in the recent VRP literature. The decades of development regarding the non-exact methods have led to an efficient template for solving the VRP revolving around **Local Search** [10]. Algorithm 1.1 summarises the local search-based methods. First, they build and evaluate an initial solution, which we are going to be referring as the **Initialisation Step**. Second, the solution is improved considering some iterative process. In this **Improvement Step** the solution is modified according to the chosen perturbation heuristics (also called neighbourhoods moves or operators). Consequently, the heuristics search for a better solution. However, it is the **Evaluation and Acceptance Step** that decides whether the changes provoked by the heuristics will be accepted or not, regardless of their quality. This step is especially important in problems which are hard to compute

[4]The non-exact approaches include heuristics and metaheuristics (and arguably hyper-heuristics in a third category, although this is not consensus, as some authors consider hyper-heuristics as a type of metaheuristic [71]). Regardless, it is common for using the word heuristic when talking about any non-exact method.

[5]Reasonable here means at most a few hours, but mostly within minutes. Exact approaches can take up to days or months of execution time, depending on the scale of the problem.

the solution value, or problems that contain several local optima. The Improvement and Evaluation and Acceptance steps are repeated until a stopping criteria is met, returning the final solution.

Algorithm 1.1 The Local Search Framework for the VRP

```
1: procedure LOCAL SEARCH(Problem Instance)
2:     s ← Initial_Solution()          } Initialisation Step
3:     while Stop Criteria do
4:         s' ← Modify_Solution(s)     } Improvement Step
5:         Evaluate(s')
6:         if Acceptance Criteria is satisfied then   } Evaluation and
7:             s ← s'                                   Acceptance Step
8:     return s
```

The quality of this final solution, therefore, depends on the starting point, which moves are used over time to modify the solution and which criteria are used to accept the modifications. A problem, therefore, arises when building a local search method: **which local search components to use to improve the method's performance?**

1.1.1 Manually Designed Decisions in Local Search

As each component in a local search method contributes in a different way to the overall search task, it becomes a challenge to select the ones that are going to be performing the best for a given scenario. The traditional heuristics might not find good solutions across different datasets and the most recent metaheuristics are increasingly complex. These most advanced methods select the components based on how they can contribute to the search, complementing each other [10]. Doing so, however, is not trivial, requiring a vast knowledge on how the VRP works and considerable experiments. Therefore, several applications just combine known neighbourhoods and components, because the developers expect that they will complement each other's weaknesses.

Such concept could be wasting resources by spending time in non-promising neighbourhoods as the local search still has to evaluate them fully, or at least partially, before going to the next one — resources that are especially important in large-scale problems (which will be further covered next in Section 1.1.2). This waste is partially due to the previous point, as these components are selected aiming to minimise a specific characteristic, whereas their viability is limited to certain scenarios. For example, the classic **2-OPT** is good for removing intersections in a route. However, without the presence of those intersections, this neighbourhood is not as effective. Hence, local search-based methods that apply the 2-OPT at all iterations/routes, might be significantly less effective in certain scenarios.

Therefore, when manually creating the more advanced methods, it requires that the components to be either carefully selected with complementing and powerful moves (as in [10] and [184]), or to accept some loss of effectiveness from components that are general enough. Even if trying to do a careful selection, it would be difficult to do so, either because of the need to specific tune methods for the given scenario or due to having little information available on how these components work on the search space, and therefore, on how to work with them more efficiently. Additionally, this type of information only recently became available and it is still deemed as difficult as shown in [137], which did a fitness landscape analysis for the VRP, including the Exchange, Relocate and 2-OPT moves.

Another point that can be taken from this is that any manually designed algorithm is sub-par. In other words, any algorithm that finds its solution based on the developer's intuition or the instance's characteristics can actually be under-performing. That is because they lack adaptation to the particularities of the solution type, the instance configuration or the search process. Additionally, the selected components might not cover the search space well enough — limiting the method's ability to find possibly outstanding solutions (or even the optimal). Some methods try to overcome these blind-spots by incorporating exact approaches

to their methods. These, however, require a deep understanding of both the problem and the exact solvers. These are called matheuristics, such as in [125,147,157,176], and they require very efficient (and often monetarily expensive) solvers, while often requiring the addition of complex mathematical pruning by experts. As our goal is to find better solutions, reducing costs and improving quality of life for everyone along the supply-chain, we want to step away from making premature decisions or limiting the application development to involve too many experts.

Rather than trying to understand each possible scenario and every solution characteristic, which can lead to better solutions, we can aim to design algorithms whose decisions and parameter values will learn and adapt to those scenarios and characteristics automatically. The algorithms would evolve themselves to the best possible fit for the given instances. Such idea might seem impossible if considering a single method for solving several problems, as the "no free lunch" theorem attests [194]: any good performance over one class of problems is counterbalanced by another class underperforming. But there have been advances in methods which aim to have less dependency on manually designed decisions, solving a wide range of problem types. This is possible due to these methods actually being "unique" for each instance, as they have components defined at execution time.

However, the automatic tuning of parameters has been considered and explored by packages like the IRACE [123]. As IRACE is a generic tool which can be used to tune parameters, it can be used to tune existing metaheuristics in order to make them more general and less dependent on arbitrary tuning or limited testing. Nonetheless, these tunings would still be dependent on the overall design of the metaheuristic considered, which might still suffer with the aforementioned issues and with the large search space.

1.1.2 Dealing with Large Search Space in Local Search

The main issue in solving the LSVRP is the computational cost of the utilised neighbourhoods in the local search Improvement Step. Although these are generally much more affordable than the exponential search space from the exact approaches, they still can have a fairly large search space, such as $\mathcal{O}(n^4)$ for the Cross-Exchange move. For the smaller instances (where $n \leq 100$), we have a large but still within millions number of possible moves, which the standard computer can perform fairly well. However, for larger instances, such a neighbourhood would be too expensive time-wise, especially across several iterations.

To deal with this issue, several methods in literature have utilised two main approaches (which are not exclusive to local search): **divide-and-conquer**, where they split the search space into smaller easier-to-solve pieces, or by **pruning the search space**, where some solutions are disregarded from search. Of course these two can also be utilised simultaneously, although this practice is not as common. Regardless of their pros and cons, both approaches require a deep understanding or experience with VRP, since any form of changing the search space might damage the overall search, i.e., if not done correctly it might remove good solutions from the pool. For example, as pointed out by [90], in a 532-city Travelling Salesperson Problem (TSP)[6] instance the optimal solution utilises the $22nd$ nearest neighbour of one of the endpoints. Therefore, any pruning method which searches for less than that will never find this solution. Although the example is for the TSP, the principle is the same for to the VRP.

Although finding the optimal solution might not be the goal of most heuristic designs, it is still desirable to have the best solutions, as these are translated to cost reductions in real applications. Because of this, such limits to the solution search space should be considered carefully. Methods

[6]The TSP is a similar problem to the VRP, although only having one "vehicle" or salesperson and no capacity constraints. The problem consists in finding a minimal-cost Hamiltonian cycle in a graph.

that have such reductions rely on the designer's expertise to set-them-up. As this is a non-trivial task, they tend to give up on some other aspects, such as the method's generalisability or for it to be too parameter dependent. This argument on generalisability goes back to what was discussed in Section 1.1.1, but in this case, the consequences can be more severe — as limiting the search space even further (considering the limitations already given by the selected local search components) may drive the search to poor solutions. It is important to note that we are not considering GPU techniques as ways to improve the handling of the large-space, as was done in [197], as these types of improvements would not handle the exponential space directly, limiting the gains up to a certain point.

1.2 Motivations

From the previous section we can identify two limitations in the development of local search methods for the LSVRP: the number of parameters and components that can be used — each taking a portion of the processing time even when not promising; and the search size which is too large — requiring additional pruning of the search space. In this section, we expand on those limitations to build our motivations. Here we go back to the Local Search framework steps: Initialisation, Improvement and Evaluation and Acceptance. These three components are often manually arranged in heuristic design, but there has been some research which attempts to automate the creation and configuration of such methods, and we will present them next, before introducing our motivations.

1.2.1 Complexity of the Heuristic Search Space

It is not a brand new idea to develop methods that search through the heuristic space in order to automatically build or configure methods for COP. This is done mainly with what is known as Hyper-Heuristic (HH)

[29] or, more recently, the General Combinatorial Optimisation Problem (GCOP)[7] [156]. HH methods are formally defined as high-level algorithms which search the heuristic space rather than solution space [28]. As pointed in [30] the HHs are a semi-automated process, where the human designs the search space and the computer designs the heuristic and how to search in such space. They aim to be generalisable across different problems, variants or instance types. Although this statement seemingly contradicts the *no free lunch* theorem already mentioned, the HHs actually develop a different configuration for each problem in hand. The resulting method will likely be unique, considering problem characteristics and available components, reducing the need for an expert to set decisions based on the problem characteristics.

HH has, therefore, the **potential** to find better solutions when compared to any manually designed method, if given the same components and moves to search for. For example, given an expert-created algorithm that utilises a specific set of local search operators and finds solution x_i for instance i. Any HH that utilises the same set of local search operators would be able to find at least the same result x_i, which would consist of using the same local search operators and in the same order as the expert has. But, the same HH would allow to find a solution x'_i better than x_i, which is a combination of the same components not considered nor tested by the expert. On top of that, a HH can also introduce new components that itself designed and were not considered by any human.

However, one challenge in using such types of algorithms to search heuristic configuration is the size of the heuristic search space in itself. Even if considering only the local search framework, there are several possible neighbourhoods, stopping criteria, acceptance criteria, initial so-

[7] The GCOPs are a much more recent methodology which is a especial case of HH. The GCOPs aim to create a standard way to build search algorithms for COPs. As their definition is still recent, with little literature presence, we refrain to utilise GCOPs in our scopes, and otherwise utilise only HH for referring to automatic solvers for COPs, even if they are similar in nature.

lution methods etc., some with their own parameters (for example, the Cross-Exchange neighbourhood has a maximum-exchange-size parameter), all with different levels of impact on the search process. In some cases, the number of possible designs with parameters can be of exponential size [156][8] — which makes this problem as hard as solving any COP. On top of that, we also have a complex solution search space where the selected heuristic configuration has to work with (in the case of the LSVRP, another COP of exponential search size of likely even larger size). These two complex layers make it challenging to find methods which perform well across distinct instances.

Hence, finding the configuration that allows to find an $x'_i \leq x_i$ is a computationally expensive task. This is corroborated by the fact that there is still a considerable gap between the state-of-the-art metaheuristics and hyper-heuristics, at least when looking at the VRP and variants[9]. This is noted in [165] and [172], where the HHs are mostly not competitive amidst the specialised metaheuristics.

When analysing current HH methods for the VRP and LSVRP (as can be seen in Section 2.5.5), most of them fail to present concerns about the scalability for larger scale problems. Although a considerable amount of algorithms for the VRP employ the template shown in Algorithm 1.1 (both

[8]Although this is shown in the reference, the scale can be easily illustrated by considering a simple Genetic Algorithm and the number of crossover operators, mutation operators and their rate parameters, which can vary from 0% to 100%. If we have a pool of only 5 different crossover operators and another 5 mutation ones (which is a rough underestimation), including only these two rate parameters we have 25000 possible configurations, considering a step of 10% — not even accounting that these rates can be changed during the learning process. If we include additional steps, such as selection, elitism etc., the number of possible configurations would be even higher.

[9]Specifically for non-dynamic problems, as dynamic ones have seen HHs been the state-of-the-art for some time. The dynamic are variants which present changes to the problem at runtime. For example, in the VRP, a dynamic variant has new customers being added or removed after the vehicles already left the depot, requiring re-optimisation of the routes.

HH-based and other metaheuristics), it is not uncommon to see that HHs are still using manually designed decisions in some of the steps. It is fairly standard for HHs to focus only on the initialisation, like the GP-based HH as in [83], or the improvement step, like in [167].

One of the reasons for that under-performance is because HHs trade quality for generalisation. In other words, a lot of HH designs aim to focus on simplicity, considering components which are simple and for general purposes. Thus, it is understandable that they perform worse than metaheuristics since the HHs utilise less expertise knowledge regarding each problem. However, considering a metaheuristic will utilise a combination of components and decision-making options based on expertise, and since this combination is within the heuristic search space, we expect HHs to at least find the same results as metaheuristics.

The recent increasing interest in the VRP, at least in number of published work [168], comes by solving the variants of the problem, such as uncertain, dynamic and electric. Since each have their own particularities, it is valid that a new technique is developed to tackle them. However, as already criticised by [187], developing a method for each possible variant seems like a waste of resources. As all variations still face the problems described in Section 1.1, i.e. the manual decisions and the large search space, we want to focus on the basic formulation, from which it can be extended for these variants.

Hence, in this thesis we want to look at these common points and look into them as stepping points for future works. We can look at the different steps involved in a local search-based metaheuristics and try to automatically optimise them, individually or as a whole, since the manually-designed methods are prone to bias and the lack of generalisation. We present why these steps are important next.

1.2.2 Initialisation

The **Initialisation Step** affects the performance of the local search-based methods (as shown in [143]), by defining the starting point on the solution search space. When looking at the recent work, most of them use a generic construction technique in the initialisation step which provides a feasible solution, such as the Clark and Wright Savings heuristic [37], the Nearest Neighbour [18], the Sweep heuristic [76, 195] and one of the pioneers to focus on large-scale, the heuristic of [158].

More recently, the work of [9] made some improvement on the Clark and Wright Savings for large-scale problems, trading up a bit of its effectiveness for efficiency. All these methods, with the exception of the Savings modification, are over 50 years old (with the exception of [158] with only 30 years, which is the least popular among these), and are still heavily used today. These methods are selected based on them being easy-to-implement or by the designer's affinity to that heuristic. This is due to being much easier to find improvement of a not-so-good solution rather than creating a very good one. Hence, this step has gotten less attention of the scientific community.

A counter argument for such concept could be: an already good solution leads to a local optimum much faster, and escaping it could be harder for the next local search steps, hence the initialisation methods that perform a worse initial cost are less bound to such traps. However, if that is the case, and given the complexity of the search space of a problem such as the VRP, it would be also equally easy to trap the worse initial solutions in their own bad local optima, not allowing for a better solution to be found given the same operators.

Given that the automatic heuristic design methods for initialisation heuristics are failing to provide an advantage in the search process, either due to the use of wrong initialisation components or by underestimating its impact (which often leads to ignoring this step). As there is no proper study to validate these, we are motivated to propose ways to do so.

1.2.3 Improvement — Operator Selection Strategies

The core phase of a local search-based heuristic, the **Improvement Step** requires a set of neighbourhoods which modify the solution. These neighbourhoods are mainly manually selected, as discussed in Section 1.1.1. Therefore, we turn to HHs to overcome this limitation. Considering one of the main types of HHs [28], a Selection HH focuses on selecting the utilised move operators, either at the decision point or by learning the order of the operators to be applied. A Selection HH strategy would avoid bias over a specific neighbourhood and let the HH learn which ones work better for the instances considered. Unlike metaheuristics, which handle the solution space more directly, the HHs only deal with the heuristic space and is only partially aware of improvements or worsening of the solution. This extra layer separating the solution to the method can become a liability (due to the addition of extra computational steps) for larger scales, where execution times become more sensitive, easier to escalate very rapidly.

Considering that a local optima for an operator's neighbourhood is often not a local optima for another one, the order in which they are applied can change the direction of the search completely. For example, if a metaheuristic method focuses on inter-route moves before intra-route moves, it could lead the number of routes and their positions to a very different place if done the other-way around. For some instances, the first-case might be a good strategy, but not for others, which can prefer the second one, or either a completely different mix of the operators.

A Selection HH approach could identify these trends and optimise this order, whether doing it online or offline. For online ones, the strategy is to select the next operator at certain times (for example, after reaching a local optima, if using a Hill Climbing strategy). For offline Selection HHs, however, they pre-determine the order of the operators and fix them throughout the execution. Although the training process to find such order can be expensive, after trained the method becomes just as fast (if not faster) than an online approach, since several online strategies might have

to calculate which operator to select at each decision point. This fixed operator order sounds less optimal, since they are not adapting to each instance being solved. This, as is in several method decisions, is a trade-off. Even if the instances are not uniform, such technique could find an average one that performs fairly well across them, avoiding having instances under-performing with a manually-set fixed strategy. And these offline approaches have additional advantages implementation-wise and interpretation-wise, as the developers can check the trained models.

In the Selection HH literature, however, there is a disregard for the search space size of each local search operator, which goes back to the problem of the large-scale not being dealt. For example, in the work of [127] a Grammatical Evolution method is proposed, but there is no mention of handling large scales. As one of their improvement options is a full metaheuristic (the ILS), we can assume that this method does will not scale well for large-scale problems.

For non-evolutionary-based methods, one pioneer HH work of [68] introduces a hill climbing HH which builds a sequence of constructive and improvement operators to build and improve routes in sequence. However, even back then, the results were still slightly worse when compared to the much more simplistic metaheuristics of the time — and also without concern for scalability. In this category, we have found one online work that handles large-scale problems. In [167] the authors introduced a two-phase HH for the LSVRP with Time-Windows (LSVRPTW, a variant where each customer has a specific time-window to be visited). But even in this case, the large scale is dealt with by splitting the instance in the early stages of the method, without considering the improvement phase later.

Other work, such as [172] finds a hybrid method, where it uses a GP to generate a set of solutions for the VRP. Followed by a Selection HH which randomly selects modifications that are going to be applied to the solutions. No concern is given to large-scale problems.

Hence, due to the lack of work here and by considering their lack of success, we investigate new Selection HH methods by also considering the scalability for large-scale problems.

1.2.4 Improvement — Operator Search Space Size

In VRP method design, it is common to have the selected neighbourhoods search for all neighbouring solutions, no matter if selected manually or automatically. This is because they are not considering the scale of the instances or because they are designed for small-scale. As expanding these to large-scale is not a trivial quest, a challenge arises.

Some methods try to prune their search for the **Improvement step**. However, the strategies selected can be just as manually designed as the rest of their methods. For example, in [118] the authors apply a Simulated Annealing (SA) for solving LSVRP instances with up to 1200 customers. Their SA limits the number of neighbours to be explored to the 40 nearest connected nodes. The work of [9] presents the Knowledge-Guided Local Search (KGLS), solving instances of size up to 30000 customers. They manually selected a few but powerful neighbourhood moves, in order to maximise effectiveness. These powerful neighbourhoods, however, have a very large search space, which was dealt with by limiting the operator to only consider the moves that connects a customer with one of its 30 closest customers. Similarly, other work such as [131], also limit some of their operators to a sub-set of closest customers.

Other methods limit the search space through fixed-clusters. For example, in [88], the authors propose a balanced K-Means to find a set of clusters to reduce the search space in large-scale instances. In a more recent work, an evolutionary multi-objective method based on route grouping was proposed in [196]. This kind of technique can be very efficient, as instead of solving a VRP with, for example, 1200 customers, they can solve 6 VRPs of size 200, a size much easier to deal with. However, the cluster-

ing strategy is very susceptible to the clusters created, especially if limiting the exchange of customers between them. Additionally, for specific customers geographical distributions or with high variation of demand, it can be nearly impossible to find efficient ways to split the customers[10].

More unique methods have also been proposed, whether by exploiting CPU/GPU parallelism, as in [182] and [197], the use of reduction strategies to improve the efficiency, as in [107] and [9] (this work adds more layers of improvement rather than just limiting the number of customers), or the use of solution-based characteristics in [121], where they compare the distance between routes before exploring the operators' moves.

These works represent the state of how the search space is considered for large-scale problems. Although there has been a recent increase in interest to solve these larger instances, they still heavily rely on manually adjusted or fixed strategies. We want to consider a method which automatically learns how to adapt to the search space of the instance being solved at runtime.

1.2.5 Evaluation and Acceptance Criteria

The **Acceptance and Evaluation step** have their own characteristics which impact the performance of the methods. The *evaluation part*, which calculates the new solution cost, can be quite expensive to compute, especially since it is done on each iteration. The traditional strategy would just check all routes length (edge by edge) and return the total cost. However, this was proven to be too slow for large scale, with $\mathcal{O}(n)$ cost. There has been

[10]This can be easily verified with an example: in a highly dense population of customers (for example, a big city); if the number of customers within a cluster has to be balanced, there will likely be several neighbours (in the actual sense of the word — street neighbours) in different clusters. Although possible, it makes more sense to allow these types of customers to be considered together in the search, something that would not happen if tight clusters are used. Unbalanced groups is also possible, but it is perhaps more challenging to do without manual input.

an effort to improve the efficiency of solution evaluation, such as in [187] where the routes are not fully calculated after moves, but rather only the parts involved in the move. This became the standard for several metaheuristics, showing to be very efficient.

On the other hand, the *acceptance part* is the polar opposite. Each metaheuristic utilises its own acceptance criteria. From the most simple ones, such as the Hill Climbing (HC) strategy, which accepts only improving moves, to a more complex one, such as the ones employed in HHs including stochastic strategies [54] or such as tabu-list, simulated-annealing-based, etc. Determining what kind of acceptance strategy can determine the effectiveness of the search process. This can be easily explained with an example: given an initial solution x that has been perturbed with operator f, resulting in x', such that $x' < x$. As this move improved the solution, the new x' can substitute the current solution ($x \leftarrow x'$). However, x' is a local optima to operator f. If in the next iteration we utilise the operator g, such scenario is possible: $g(x) = x''$, where x'' is equal to the initial x. If such move is accepted, and no other operators are given, this method might enter an endless loop. Although this is a very simplistic example, it shows the traps that are possible if this step is not well crafted.

When it comes to large scale, there is no obvious specific modification in that step that can be done to improve its efficiency directly, as this would require the landscape of the search space to be known *a priori*. However, the implications of a non-effective strategy can be seen as a loss of efficiency, since they would go through more solutions, using expensive improvement operators (expensive due to the scale) to find new solutions. Among the most recent strategies, the KGLS [9] uses a strategy which changes the evaluation function, plus three different penalisation criteria that are used alternatively, to make stuck solutions cost more than they actually do. Those penalisation modes are based on the cost of the edges, their width or both, hence, the method is aiming to use a "smart" strategy, since it is penalising the routes that are more costly.

We focus on this work due to it being one of the state-of-the-art metaheuristics for the LSVRP, especially since it is entirely built on a local search framework. However, we question whether a fixed penalisation criteria is the most effective way to utilise the solution characteristics. We argue that we can optimise this process by learning when to apply each mode — since different instances present different characteristics.

1.3 Research Goals

The main goal of this thesis is to develop **a new Automatic algorithm design approach** to the Large-Scale Vehicle Routing Problem, as in automatically designed heuristics and/or combined with learning methods which minimises the input of expert-based decisions. The approaches need to be **efficient**, i.e. scaling well for a larger number of customers, and show **generalisation** on different instance types and configurations.

We approach this goal by considering applying automatic decision processes in all main steps of the local search framework: Initialisation, Improvement and Acceptance. Although these steps are dependent on each other, they will have their own contributions to each respective step.

Our overall goal can be divided into theses four key objectives:

1. *Learning Effective Initialisation*

   We ask a couple of vital questions that allow us to understand and improve the initialisation step as a whole:

   How much impact does an initial solution of a local search-based metaheuristic have? And can we learn to utilise this solution in favour of our search?

   The overall idea in literature is that, given a Large-Scale VRP instance, any search method becomes more sensitive to the starting point, because of the extra amount of time required to improve it.

However, we argue that it is not only the solution cost which affects that performance, but also the initial solution's characteristics. Although better starting points are more likely to lead to a better local optima, these points might also be harder to escape. Hence we propose an experimental study to show whether worse-performing heuristics can provide a better final solution, and in which conditions. We also learn to utilise these results to build a classifier to predict better performing methods, and to utilise some characteristics to build new solutions that lead to better improvement. Hence, our first objective in this thesis is to **develop automatic initialisation methods which improves the effectiveness and efficiency of a given local search-based metaheuristics for the LSVRP**.

2. *Operator Selection and Bounds with Evolutionary Hyper-Heuristics*

   Considering Selection hyper-heuristic methods and their lack of consideration for large-scale instances, we aim to answer the following research questions:

   Does the order of the neighbourhoods operators affect the search? If so, can we optimise it? And can we automatically incorporate proper limits to their neighbourhood scope without reducing their effectiveness?

   In the improvement step, the neighbourhood operators represent the most impactful decision when designing new methods. The utilisation of these operators affect the effectiveness and also efficiency of the method. On top of that, when dealing with large-scale, reducing the search space size is a tough challenge in COPs, as these reductions can remove good solutions from the pool. Here we explore the use of Genetic Algorithm (GA) as a HH to optimise the order and the size of these components. We introduce new chromosomes which can be used to automatically define a new local search improvement step and how much to limit the solution search space,

using the evolution pressure to determine the safe limits and the best order of operands. The second objective of this thesis is **to develop Selection HHs using a genetic algorithm with new chromosomes which incorporate a local search-based method improvement step, including efficient search space limits for the LSVRP.**

3. *Learning to Guide Solution Search Space*

   Considering that most metaheuristics utilise a fixed search space strategy, we question:

   Can we limit the neighbourhood search space for each operator adaptively? Does that improve the efficiency? Does it come with a loss of effectiveness?

   When dealing with the large search space, most methods apply some sort of divide-and-conquer strategy directly to the customer space. While some work limit the operators' scope by a fixed limit, we debate whether these strategies are the most effective in improving the efficiency. We analyse the behaviour of the search space for a traditional fixed strategy to observe if these limits are similar for different neighbourhoods. We then compare this fixed operator limit with an adaptive one, which tries to optimise the search in an online way. Hence, in the third objective we introduce a **new strategy that adaptively changes the search limits for each inter-route operator when solving the LSVRP.**

4. *Learning Acceptance Strategies to Escape from Local Optima*

   Getting stuck in local optima is one of the main challenges of local search-based methods. As most metaheuristic strategies have been using manually designed decisions to jump out of local optima, we drive this objective with the following questions:

   Can we learn to automatically select the acceptance criteria? Is having a pool of criteria beneficial? Can we automatically learn a new

method to drive out of local optima?

A large search space also comes with several local optima. Because of that, this step of a local search framework can become even more sensitive to the fluctuations on the solution landscape across different instances. The traditional methods, however, do not differentiate utilising the different strategies available, and even most hyper-heuristics do not consider this part of the process. We want to overcome this by analysing the impacts of different strategies under the same framework. We utilise ML methods to learn when these strategies will be more efficient and how to build new ones. Therefore, for the 4th and final objective we **introduce the use of ML techniques to differentiate the most-suitable acceptance criteria and to build novel ones, increasing the effectiveness in the search for better LSVRP solutions.**

1.4 Major Contributions

This thesis makes the following scientific contributions:

1. This thesis shows an analysis of the impact of an initial solution to the local search performance. We present arguments that the initialisation process can be relevant and should be considered more carefully, as they can direct the search to a better space more quickly — even with the same operators. We achieve this by running the same local search-based method initialised by different constructive heuristics. We utilise the information extracted to build a machine learning (ML) model which utilises the solutions characteristics as input and outputs the predicted best heuristic, producing better results than a fixed strategy approach. We also introduce the use of Genetic Programming Hyper-Heuristic (GPHH) to utilise solution characteristics to build initial routes. Our results show that even with

simple route construction steps, we are able to initialise a solution in a specific region of the search space which provides faster and better improvement than those initialised by traditional methods.

Parts of this contribution have been published in:

Joao Guilherme Cavalcanti Costa, Yi Mei, Mengjie Zhang. Learning Initialisation Heuristic for Large Scale Vehicle Routing Problem with Genetic Programming. IEEE Congress on Evolutionary Computation (CEC 2021) (pp. 1864-1871). IEEE

Joao Guilherme Cavalcanti Costa, Yi Mei, Mengjie Zhang. Learning to Select Initialisation Heuristic for Vehicle Routing Problem. (Accepted by GECCO 2023)

2. This thesis shows how effective and efficient Hyper-Heuristics strategies can be achieved by evolving the operator order and their limits simultaneously. We achieve this by considering a Selection HH using a Genetic Algorithm as the base HH framework, and by introducing two new distinct chromosomes and a new embedded clustering method. The chromosomes have the following attributes: the first one has two layers — one for the operator to be used and another one for limiting the search space of said operator; the second chromosome adds a third layer to further limit the search space, but also adds a few extra alleles to select the initialisation heuristic and the penalisation functions for a Guided Local Search framework. The clustering technique utilises a classic chromosome but we introduce a new form of evaluating it — where we separate the intra-route and inter-route operators, adjusting the initial clusters based on the inter-route moves, guiding the space towards the routes changes. Our results show that the use of the Selection HHs and the new chromosomes and non-manual limits to the search space were able to increase the efficiency and effectiveness across different instances and even under different local search frameworks.

Parts of this contribution have been published in:

Joao Guilherme Cavalcanti Costa, Yi Mei, Mengjie Zhang. Cluster-based Hyper-Heuristic for Large-Scale Vehicle Routing Problem. In 2020 IEEE Congress on Evolutionary Computation (CEC 2020) (pp. 1-8). IEEE

Joao Guilherme Cavalcanti Costa, Yi Mei, Mengjie Zhang. Adaptive Search Space through Evolutionary Hyper-Heuristics for the Large-Scale Vehicle Routing Problem. IEEE Symposium Series on Computational Intelligence (SSCI 2020) (pp. 2415-2422). IEEE

Joao Guilherme Cavalcanti Costa, Yi Mei, Mengjie Zhang. An Evolutionary Hyper-Heuristic Approach to the Large Scale Vehicle Routing Problem. IEEE Congress on Evolutionary Computation (CEC 2021) (pp. 2109-2116). IEEE

3. This thesis shows the use of adaptive strategies to learn how to limit the search space at runtime. We achieve this by introducing a new stochastic heuristic which attempts to learn the appropriate size of the search for each inter-route operator. These limits are set based on a number of closest customers that can be explored during the search. We measure the worst-case scenario in which we observe the customers that have the best improving move is usually far below the upper limit, hence we apply an strategy to reduce such limit. If new solutions are not found, the limit automatically increases again, allowing for a flexible and efficient search. Our results show that doing this allows a significant increase in the number of iterations that can be performed in the same time-frame, and better solution can be found due to these extra iterations.

   Parts of this contribution have been published in:

   Joao Guilherme Cavalcanti Costa, Yi Mei, Mengjie Zhang. Guided local search with an adaptive neighbourhood size heuristic for large

scale vehicle routing problems. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2022) (pp. 213-221). ACM

4. This thesis shows an analysis of the impact of the penalisation criteria to a Guided Local Search framework. We show that each instance has a preferred penalisation mode which can provide a significant change in the solution quality. We do this by running the same GLS method and fixing the penalisation strategy. We utilise the solutions characteristics to build two ML models which predict the best penalisation mode, using a classification and a regression model. Both were able to estimate the best mode, resulting in better results when compared to a fixed penalisation strategy.

   Parts of this contribution have been published in:

   Joao Guilherme Cavalcanti Costa, Yi Mei, Mengjie Zhang. Learning Penalisation Criterion of Guided Local Search for Large Scale Vehicle Routing Problem. IEEE Symposium Series on Computational Intelligence (SSCI 2021) (pp. 1-8). IEEE

1.5 Organisation of Thesis

The remainder of this thesis is organised as follows. Chapter 2 presents some basic concepts used in this thesis, as well as the literature review on the Vehicle Routing Problem and the methods used for solving it. In Chapters 3-6 we present the main contributions of this thesis. We end each contribution chapter by giving a brief discussion on scalability and computational costs. Finally, Chapter 7 concludes this thesis and presents potential future work. The main chapters are detailed next. An overview of all chapters distribution is given in Figure 1.1.

Chapter 2 presents an overhaul on the basic concepts discussed in this thesis, including neighbourhood, local search and heuristic methods. Then,

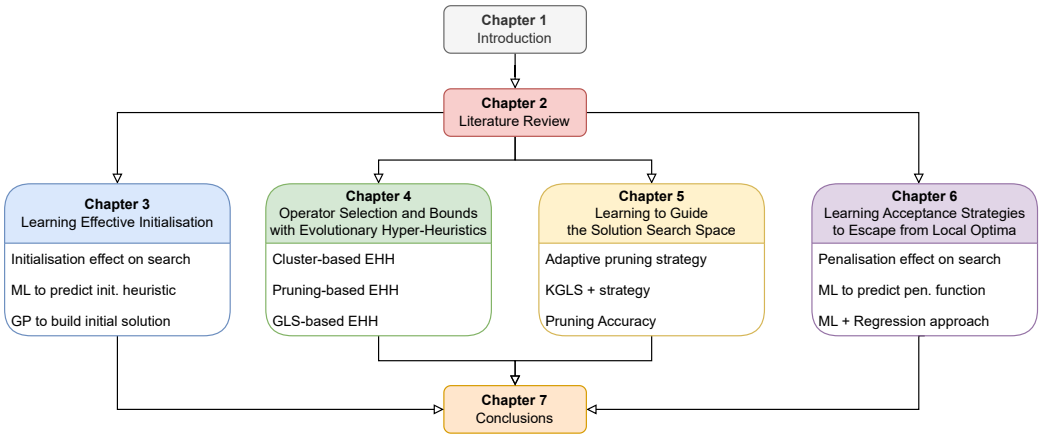


Figure 1.1: Outline of thesis chapters and main contributions.

an overview of the relevant work in both Large-Scale Vehicle Routing Problems and Hyper-Heuristic for VRP are detailed.

Chapter 3 describes our first contribution considering the initialisation step. We present a study on the impact of the initial solution to the search and develop two methods to improve the quality of LSVRP solving. The first one utilises ML methods to predict which existing initialisation heuristic will be more effective. The second method utilises GPHH to create initial solutions in specific regions of the search space from which it is easier to find improvement.

Chapter 4 explores the use of Selection HH to improve the order of and limit the operators used in the improvement stage of a local search-based heuristic. We present three different strategies, all based on Genetic Algorithms, each with its own novel genetic components, such as chromosome or decoding scheme.

Chapter 5 contributes to the automatic pruning of the search space in an online way. Here we present our method which adaptively learns each operator limits, considering their performance on the previous iteration.

Chapter 6 describes our strategy to predict the penalisation criteria to be used in the KGLS context. We introduce labels to the data and ap-

ply several ML methods modelled as classifier and regressor, including a novel GP method, to predict the class or value on unseen instances.

Finally, Chapter 7 presents the overall conclusions from each chapter and the thesis. Then, a future prospect of possible next steps is given.

Chapter 2

Literature Review

In this chapter we first introduce some basic concepts that are used throughout this thesis. We also present some of the related work regarding the Vehicle Routing Problem and automated approaches.

2.1 Basic Concepts

In this section, we present and briefly explain some basic concepts of Optimisation problems and related concepts that are used in this thesis. We are following the description order based on the books of [143] and [117], where most of the chosen definitions are mathematically defined. Additionally we present some descriptions on search emphasis and types of methods, which we judged to be important concepts that needed to be added and are based on other references. These general concepts are fairly common in the optimisation field and are going to be used throughout this thesis.

2.1.1 Combinatorial Optimisation Problem

Combinatorial optimisation is a field of study situated between computer science, mathematics and operations research, where a set of **decision**

making problems (usually $\mathcal{NP}$-hard) containing discrete options, i.e. has **discrete variables**, are studied and solved [1]. A combinatorial optimisation **problem** (COP) is defined as a problem which contains a set of discrete variables, constraints and either minimisation or maximisation function as cost. Additionally, the number of possible solutions is usually so large that an exhaustive search is too costly.

Formally (based on [117, 143]) an instance of a COP is defined by the mapping: $f : S \to \mathbb{R}$, wherein the pair (S, f), S is the domain set with feasible points, and f is a cost function. The problem is to find $x \in S$ for which: $f(x) \leq f(y)$ for all $y \in S$, given a minimisation function[1]. This x is known as the global optimum, since no solution in S has a lower cost, and is usually denoted as x^*, with f^* being the optimal cost of function f when x^* is applied.

2.1.2 Neighbourhood

A neighbourhood is composed of a set of solutions that are nearby in some sense, given a specific problem [143]. In other words, given $x \in S$ the neighbourhood set $N(x)$ contains the solutions similar to x in some aspect. Formally, a neighbourhood N is defined as [143]: given the pair (S, f), the mapping function $N : S \to 2^S$.

Briefly explaining, what a neighbourhood means is that any solution x which can be transformed with a specific modification function. For example, given the list of integers $x = (1, 2, 3, 4)$, then some of the neighbours of x according to a simple swap function (which swaps between two given positions in the array) are $y = (1, 2, 4, 3)$ or $z = (2, 1, 3, 4)$. The set composed by y, z and the other swap possibilities starting from x, is the neighbourhood of x given the swap function ($N(x)$). However, a $w = (2, 1, 4, 3)$ is a neighbour of both y and z but not a neighbour of x (again, considering

[1]The standard for any optimisation problem is to be a minimisation problem, and any maximisation problem can be easily converted.

N as the swap function), since you cannot reach w from x with a single swap. Although this example might be simple, the neighbourhoods can be quite complex and not even symmetric (x can be the neighbour of y, but not the other way around, given some neighbourhood N).

A larger neighbourhood, therefore, is more probable of containing better local optima since it covers more possibilities of solutions [143]. However, they will take longer to be searched on, while smaller neighbourhoods are faster.

2.1.3 Local and Global Optima

A local optimum is the solution with the best value of the cost function f in a given neighbourhood N, given a value (or set of values) x. Formally: a solution $x \in S$ is a local optimum if $f(x) \leq f(y)$ for all $y \in N(x)$, considering a minimisation problem.

The global optimum happens when the inequality holds true for every N. Or, as formally defined by [1], considering $\hat{S}$ as the set of locally optimal solutions, the neighbourhood N is **exact**, i.e. is globally optimal, if $\hat{S} \subseteq S^*$. In simpler terms, the global optimum is no worse than any other solution in the complete solution space.

2.1.4 Local Search

Local Search is a type of heuristic approach which explores solutions in neighbourhoods starting from an initial solution. Since there is no limit to the number or shape of these neighbourhoods, local search success depends on the types of neighbourhoods applied when solving a COP. The biggest advantage comes from the speed of exploring these neighbourhoods. As put by [1]:

> 'Local search provides a robust approach to obtain high-quality solutions to problems of a realistic size in reasonable time.'

In the definition of [143], given the initial solution, a local search algorithm will explore all (or only part) of the neighbouring solutions until a local optimum is found. The neighbourhood can be changed and this process can be repeated until satisfactory improvement has been reached, Algorithm 2.1 shows this process. The choice of neighbourhood is usually based on intuition since not much theoretical information is available and is some form of art, as put by [143]. Although this definition might be outdated, there still some truth in it. As put by two recent works, [10] which shows that complementing neighbourhoods work more efficiently and in [184] who selects a new neighbourhood for their local search based on new evidence, there is now more information available, but this information is still challenging and yet to be formalised.

On the other hand, the complexity of the neighbourhood may affect the speed of the algorithm. For example, the simplex algorithm [105] can have an exponential number of steps to find the global optimal solution, but it also uses the **exact** neighbourhood for the problem. Usually, the more complex the neighbourhoods, the more likely they are to find promising results, but are slower to compute. There is a trade-off between the complexity of the neighbourhood structure and speed of search. The good local search algorithms will find a balance between these two.

Algorithm 2.1 Local Search Algorithm

```
1: procedure LOCAL SEARCH(Problem Instance)
2:     s ← Initial_Solution()
3:     while Stopping Criteria do
4:         Choose a Neighbourhood
5:         s ← Explore_Chosen_Neighbourhood(s)
6:     return s
```

How the neighbourhoods are searched represents the next main design decision of a Local Search algorithm. For example, *iterative improvement* (also known as *first improvement*) is a known way of searching a neighbourhood, where at each improvement the current solution is changed. The *de-*

scent method (or *steepest descent*) will search for the whole neighbourhood and change the solution to the best one found. A lot of heuristics apply a local search algorithm with a different search method, where they apply some unique stopping criteria, such as Tabu-Search [79], which searches the neighbourhoods based on a *tabu* list avoiding repeated moves.

Determining these decision of **which neighbourhoods** to use, in **which order** and **how to search** them are the main decisions for proposing a local search algorithm for COPs. There also seems to be a correlation between the starting point and the quality of the local optima [143]. Therefore, the initial solution is another important decision. It is from that point that the Local Search procedure starts to look for better solutions. Finding a balance between the exploration and exploitation of the neighbourhoods are fundamental to have good solutions in a viable time.

2.1.5 Exploration vs Exploitation

The two cornerstones of search, Exploration is concerned for looking for new regions, while Exploitation visit these regions more carefully through neighbourhoods [44,57]. These two concepts regard search emphasis similar to **Diversification** and **Intensification**, respectively. In more elaborated heuristics (and other non-exact methods), these concepts are applied in some form of strategy, aiming to find a balance between both.

In general, the longer heuristics run, the better solution they will find [25]. However, this trade-off of time versus quality needs to be questioned, based on what is important as a result. For example, a large investment in diversification strategies will allow for finding all kinds of regions, including the possible best one. But if little to no emphasis is given to exploitation, the optimal region can be found but the best solution might not. On the other hand, if the method invests a lot to intensify the search in a given region, then searching for different regions will take a long time. Finding this balance is one of the main goals of specialised search methods. The

design of techniques needs to take this into account.

To bring these concepts closer to the previously introduced concepts, we can consider a local search method which aims to solve the VRP. In the regular heuristic design process, a series of neighbourhoods are selected to be used such as 2-OPT, 3-OPT, Cross-Exchange, etc. How far do you explore each neighbourhood is an intensification decision, whether until the local optimum is reached or just based on some number of iterations (or any other stopping criteria). In other words, the decision is: given the current configuration of the routes, how much time we invest in finding nearby moves which can improve our solution given a 2-OPT neighbourhood. The decision on which neighbourhood to look for next is part of the diversification strategy: should the search focus on a inter-route neighbourhood such as Cross-Exchange? Or should it look for more intra-route moves?

Although this example is simple, it is probably clear to the reader now that finding this balance is not straightforward. A lot of decisions need to be made when designing a method such as the pool of neighbourhoods to be considered, the stopping criteria given to each neighbourhood, where to go next with the search.

2.1.6 Exact methods vs Heuristic vs MetaHeuristic vs Hyper-Heuristic

These terms are used a lot throughout this thesis. Here we specify each and what are their meaning in our context.

Exact methods are the approaches which not only utilise mathematical-based formulations, but also do so in a way that the final solution is the proven optimal. Most exact methods will enumerate all the possible solutions (even if implicitly). Doing this requires an enormous amount of effort, resulting in very slow methods for several scenarios. However, these methods are very important for discovering how the structure of

the problems are in reality, with their results serving as a baseline for non-exact methods. Additionally, the mathematical formulations are usually used as base for understanding a problem, allowing for other methods to be developed following a single and universal definition.

According to [166], **Heuristics** are a type of informed search strategy. In other words, they mainly utilise some problem-specific knowledge to find solutions. Historically, heuristics are the simple methods which apply iterative steps and have no way of leaving local optima, as most heuristics follow a deterministic function, i.e. producing the same result if given the same input. In this thesis we use these concepts interchangeably, both referring to non-exact search methods (metaheuristics and hyper-heuristics are heuristics in this definition), but also to the methods which apply no form of escaping local-optima traditionally used, such as constructive and improvement methods.

Metaheuristics are, as put by [71] in the *Handbook of Metaheuristics*:

> '(...) solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space.'

Therefore, metaheuristics differ from heuristics by having mechanisms that allow the escape of local-optima. Also unlike heuristics, metaheuristics often incorporate a stochastic component, producing a random outcome which works as an exploration strategy.

Hyper-Heuristics (HHs) are methods that aim to automate the design and adaptation of non-exact methods [28,29]. Although having some similarities with metaheuristics, HHs goal is to produce a heuristic which then provide the solution, as they operate on the space of heuristics and not solutions [109]. The resulting heuristic will be in charge of delivering the final solution. More recently, in [156] (including authors from [28]) have proposed a new COP where the problem is to select the components which

solve other problems. Their General COP (GCOP) considers automatic algorithm configuration, selection and composition as variables, where the goal is to optimise the resulting algorithm very much like what HH proposes. Since HH can be considered for more types of problems and their definitions involve only problems with discrete variables, it is likely to become a new standard in upcoming years for automatic algorithm design and Hyper-Heuristics development.

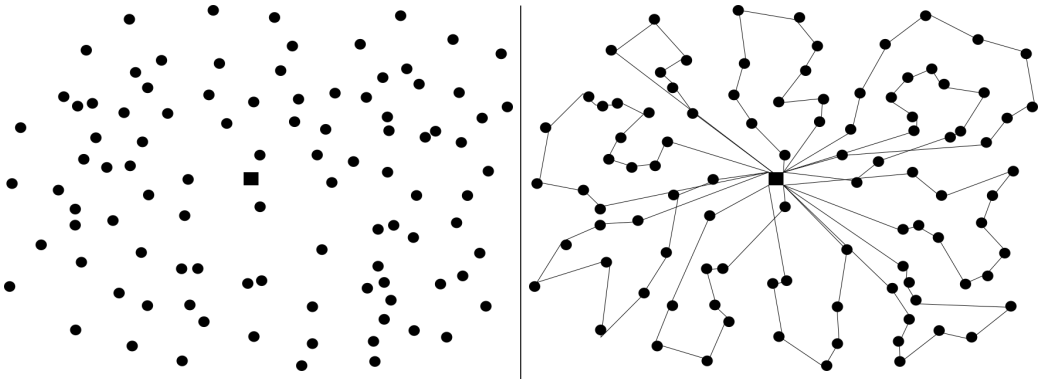
2.2 Vehicle Routing Problem

In this section we describe the main problem studied in this thesis, the VRP, and the large scale "variant". We begin by giving the overall and formal definitions for this COP. We present the main existing methods for solving it in Section 2.5.

The VRP was first described by [46], where a fuel delivery problem was studied. The original problem consisted in a fleet of trucks which needed to serve several stations, respecting the capacity of each truck. The problem has several parallels with the Travelling Salesperson[2] Problem (TSP) (in fact, the VRP is a generalisation of the TSP [153]), as in every customer (station) needs to be visited exactly once and the goal is to find the route with minimum cost, traditionally the distance travelled. The main differences from the original work from [46] are the number of vehicles (multiple instead of a single salesperson), and the added capacity of each vehicle. Figure 2.1 shows an example of VRP. The VRP is one of the most important and studied problems in the combinatorial optimisation field, with practical relevance and very difficult to solve [181]. The VRP can be applied in the industry for finding routes for transporting produce and raw materials [20,21], moving products inside or between storehouses and ports [61], or delivering to end customers. It has applications in mail [99] and gro-

[2]Originally and often called as Salesman. Salesperson is a more recent inclusive terminology.

ceries delivery [174], waste collection [84, 108, 136], recycling [102], and within health applications [32, 192]. More recently, the VRP was used in a study to protect people in vulnerable situations due to a pandemic [26]. The VRP has several variants, based on additional constraints which can make the problem more realistic to different scenarios, such as the VRP with Time-Windows (VRPTW) [169], where each customer has a specific time-window to be served, the VRP with Multiple-Depots (VRPMD) [163], where several depots are available as starting and end points, the Split-Delivery VRP (SDVRP) [7], where customers can have their delivery demands served by multiple vehicles or, more recently, the VRP with Drones (VRPD) [41, 53], where drones work in tandem with the vehicles to serve multiple customers simultaneously, among many other variants and combinations. The classical variant was later labelled as the CVRP [22], or Capacitated Vehicle Routing Problem.



(a) Instance (b) Solution

Figure 2.1: Example of how the VRP solution looks like. A set of vertexes are visited in some sequence.

The VRP (and so do other variants) is $\mathcal{NP}$-hard [117], and can be defined as the problem of determining the optimal set of routes which serve a set of customers, performed by a fleet of vehicles [181]. Formally, the VRP is defined as, based on [111]: let a directed graph $G = (V, A)$, where

V is the vertex set and A is the arc set. The vertex set represents the customers to be visited, i.e. $V = 0, 1, 2, ..., n$, and generally the 0 represents the depot point. The arc set represents the links between the vertices, $A = (i, j) : i, j \in V, i \neq j$, and are associated with a cost c_{ij}, which in the classical formulation is the distance between the two vertices i and j. If the arcs are symmetric, i.e. $a_{ij} = a_{ji}$, then they are called as edges (thus the set is called as Edge set). A set of homogeneous vehicles M with equal capacity Q is used to serve the customers. Each customer i has a demand q_i that needs to be served, and can be visited exactly once by only one vehicle. All routes start and end at the depot. The goal is to find the routes which respect all these constraints, minimising costs.

The classical mathematical formulation for the VRP based on [113,114], is such that:

$$Min \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{2.1}$$

$$\text{subject to } \sum_{j=1}^{n} x_{0j} = 2m \tag{2.2}$$

$$\sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2 \qquad (k \in V0) \tag{2.3}$$

$$\sum_{i,j\in S} x_{ij} \leq |S| - v(S) \qquad (S \subseteq V0) \tag{2.4}$$

$$x_{0j} = 0, 1, 2 \qquad (j \in V0) \tag{2.5}$$

$$x_{ij} = 0, 1 \qquad (i, j \in V0) \tag{2.6}$$

The objective function 2.1 is used to find the minimal cost routes, respecting the following constraints. Constraints 2.2 and 2.3 are degree constraints, with m being a constant or a variable, specifying the degree of each vertex. Constraints 2.4 eliminate subtours that are not fully connected and enforce the capacity restrictions, since $v(S)$ is a lower bound on the number of vehicles needed to serve all customers in S (S is a subset of V). The discrete variables x_{ij} represent the number of times the arc (i, j)

is traversed, being 2 only in the special case where a vehicle serves exactly one customer and return to the depot. The original works in [113,114] can be checked for the proofs and more details on this model. Other formulations have been proposed, more in [111], but they all present some discrete decision variable such as x here.

Overall, the VRP is an important problem that can help save money, by reducing costs, time, by reducing the route length, and even aspects such as air quality, since not only the vehicles spending less time on the streets, but also because it is possible to minimise the use of vehicles as well.

2.2.1 Large-Scale Vehicle Routing Problem

Based on the classical CVRP described above, the Large-Scale Vehicle Routing Problem (LSVRP[3]) can be considered another variant of the VRP since it needs to account for new methodologies perspectives. Although it is not a rule, and this can likely change within a few years, to be considered large scale the VRP instance needs to have at least 200 customers [9,75,95,167].

The challenges arise from the sheer size of the number of possible solutions. Additional precautions need to be taken in almost every aspect, since the traditional heuristics utilised for solving the problem, fail to keep the good performance. Simple neighbourhoods which are usually used in the VRP such as 2-Opt and 3-Opt, which have complexities of $O(n^2)$ and $O(n^3)$, respectively, will become expensive operations when n becomes large. These operators will start to become a slow-down factor in the search process. Even if assuming the increase is from a few milliseconds to a few seconds, the number of times these operators are used to search for better solutions during an optimisation process can be huge. Hence, the ripple effect will be notorious. In such conditions the traditional VRP methods will show themselves having performance issues and become unreliable for several practical applications. Therefore, when dealing with

[3]Sometimes referred as LSCVRP

large scale new ways of approaching the problem are needed.

The most well known techniques to reduce the search space for large scale are based on **divide-and-conquer**. Divide-and-conquer is a problem solving approach where a bigger problem is broken into smaller subproblems which can be solved more easily, and then a new solution is formed when combining them. For example, clustering approaches are fairly common with VRP methods, such as in [133] [199] and [38], where the customers are grouped into subsets from where a solution can be searched from this smaller space, or use them to limit the search to some limited areas.

Another approach which is fairly common when solving the VRP and variants is related to limiting the search space based on distance. In a given neighbourhood the number of compared moves depends on some metric, mostly distance-based. For example, when comparing an exchange neighbourhood which swaps customers between two distinct routes, the naive approach would compare every possible customer for this move. However, it is unlikely that some customer very far away from the current route will improve the overall distance. Although this can have drawbacks, as pointed in [90], it can be very successful, such as in [9].

The complexity of neighbourhoods is not the only issue when solving large scale problems. The developer needs to take into account other issues that are usually not present in smaller instances, such as a high memory consumption, and other optimisation techniques are usually required as well. For example, parallelisation is a quite common approach, focusing on exploiting hardware characteristics to improve the search time, such as in [182]. These topics, however, are out of the scope of this thesis.

Large-scale problems have several applications that are perhaps even more easy to occur than that of the regular VRP. If considering a garbage collection task, for example, even a small city would easily have thousands of streets/points of collection. Companies that operate at national level, such as mailing [99], or that have to serve several customers regularly, like

in soft-drink distribution [47, 48], or even in disaster relief applications [188]. The LSVRP is a relevant application that justifies its study.

2.3 Hyper-Heuristics

In this section we further explain what Hyper-Heuristics are and how they are classified. We briefly discuss how machine learning (ML) can be used in a hyper-heuristic.

As a counterpoint to the metaheuristics, Hyper-Heuristics are heuristics to choose heuristics aiming to reduce the difficulties of applying existing heuristics methods to new problems or unseen instances [28]. HHs methods are unlikely to ever beat hand-crafted, CPU-intensive problem-specific methods [165], but this can be upset by the more general design aspect. They also aim to reduce parameter and configuration dependency, which is a common problem in metaheuristics approaches. HHs can be seen as high-level methods which utilise low-level heuristics (LLH) as components, by finding an appropriate combination of these components to solve a search problem [28]. In a recent review of this definition in [31], the authors state that HHs are in the bound between ML and optimisation.

Hyper-Heuristics are methods which offer a possible way for adjusting the approach according to the problem or characteristics of the instances, without having to manually update the method for doing so. However for a HH to be successful it needs to take into account some design features such as (from [165]): the type of information available (online or offline?), the sets of heuristics to be chosen in the underlying levels should be complimentary, and since the evaluation of every solution might be expensive, if possible consider only subsets. These features influence the design decisions which usually follow some patterns. We explain how these patterns are used to classify the HHs in the next section, but first, we present a formal definition of Hyper-Heuristics.

As formally specified by [150], a HH framework searches in the heuris-

tic space H a configuration $h \in H$, which is optimal for a given problem, denoted as $h*$. The quality of this solution is measured by a function $F(h) \rightarrow R$, being formally defined as:

$$F(h*|h* \rightarrow s*, h* \in H) \leftarrow f(s*, s* \in S) = min\{f(s), s \in S\} \tag{2.7}$$

In other words, a Hyper-Heuristic goal is to find the optimal heuristic configuration $h*, h* \in H$ where the low-level functions $f(s) \rightarrow R$ are mapped into the solution space S, and is optimal regarding S (in this case considering a minimisation problem without loss of generalisation).

2.3.1 Classification of Hyper-Heuristics

According to Burke et al. [28], the HH can be classified by two different natures: nature of the heuristic search space, and the nature of the feedback type used, illustrated in Figure 2.2.

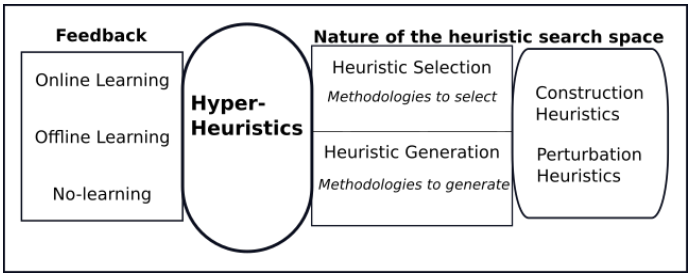


Figure 2.2: Classification of Hyper-Heuristics according to [28].

When looking at the nature of the heuristic search space, it can be divided into two main types, **selection** and **generation**. The first type is based on selecting existing heuristics and applying to the problem solution. The second type generates new heuristics based on the components of existing ones. These can also be subdivided based on which type of components they search or use, **perturbation heuristics**, where existing solutions are modified based on a neighbourhood move, or **construction**

heuristics, where solutions are built step-by-step; a mix of those is also viable. Hence, there are 4 types of possible combinations, if not considering mixed approaches.

The feedback classification is related to the learning process, if any. The learning can be **Online**, **Offline** or, in the case no learning process is applied, No-Learning. When there is Online learning, the HH method will learn while solving the problem. In other words, the method learns with the given instance of the problem and adapts the high-level strategy to determine the heuristic to be used [29]. For offline, the learning approach will happen in different stages. First, the problem instances are split into training and test sets, and are fed to the method in this training period. Then, the knowledge is gathered by the method, aiming to generalise to unseen instances [29].

2.3.2 Machine Learning as HH

The use of ML methods in combinatorial optimisation is not new, but only recently it started taking some spotlight. Some methods have used ML techniques to solve the VRP (or other COP) directly, like in [138] which uses Reinforcement Learning (RL) to solve VRP. In [191] a Deep RL is used, as well a survey on RL methods for solving transportation problems (TSP and VRP). Neural Networks were also used to solve VRP as in [94] and in [81], which also utilises fuzzy systems, or in [60], where they utilise fuzzy systems to a clustering technique for the VRP. Even some guidelines have been recently published in the use of ML with VRP in [2], and for COP in general [19], also presenting a survey on ML applied to COPs.

However, there is also the utilisation of these techniques as assistants to the problem-solving methods, mainly metaheuristics. In this case, these ML work more as HHs, since they are used to optimise the heuristic space. The benefits of this approach are not only linked to the philosophy of the HHs, as they are independent of the solution search space, but also the

use of robust and well-known techniques to automate some hard tasks on the design process. A good example comes from the iRace package, from [123], which applies ML as a way to find automatic algorithm configuration for several types of problems. Or in [203], where they utilise Support-Vector Machines (SVM) in order to assist in the parameter settings. A recent review of ML-assisted methods for VRP is given in [12].

2.4 Evolutionary Algorithms

Evolutionary Algorithms (EA) are a group of techniques inspired by Darwin's evolutionary process, where a population of individuals compete for some resources causing natural selection, or in other words, the survival of the fittest [58,175]. In EA a group of solutions evolve over time to better solve a problem. Although there are different techniques, they all work similarly and can be summarised in an algorithmic way, as shown by [132], and can be seen in Algorithm 2.2. The main idea behind EA revolves around a population P which consists of individuals that can be evaluated, being either a set of functions which are used as a problem solver or a solution to a single function. This population will then be fed to an evolutionary process, which selects individuals from the population, modify them in some way (generating the children or offspring using genetic operators) and are re-evaluated to be added back in the population. Each individual in the population is represented with a chromosome, usually an array of numbers or a tree-like structure. Each number or node is a gene of this chromosome. The modification step is usually achieved through applying a **crossover**, which is the process of trading genes between parents producing a new offspring, and **mutation**, which is the process of modifying one or more genes [161]. We briefly expand on two EA methods which are being used in this thesis: **Genetic Algorithm** (GA) and **Genetic Programming** (GP).

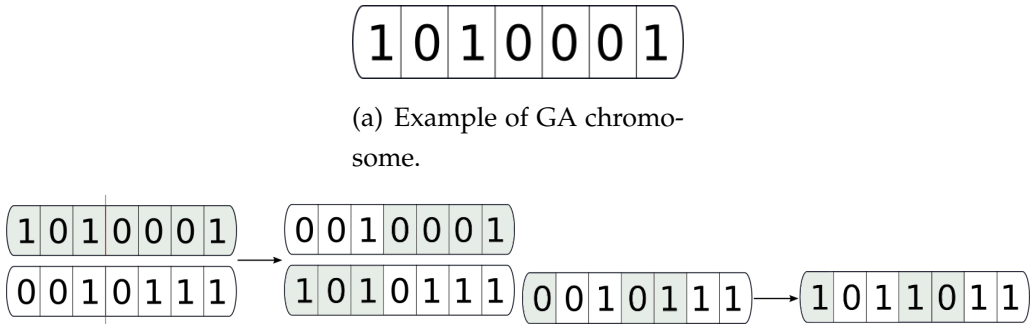
The traditional GA consists of 4 main phases given an initial popula-

Algorithm 2.2 Evolutionary Algorithm Overall Scheme

```
1: procedure EVOLUTIONARY ALGORITHM
2:     Initialise(P)
3:     Evaluate(P)
4:     while Stop Criteria do
5:         Select(P)
6:         Breeding(P)
7:         Evaluate(P)
8:     return Best(P)
```

tion: evaluation of the individuals in the population; selection of parents; application of crossover and mutation operators; and replacement of old population [153]. To exemplify each phase we will show the most common GA representation which is the binary vector, as can be seen in Figure 2.3 (a). What the genes represent vary from each application, but to illustrate let us assume we are using a GA to solve a Knapsack problem. In this problem a number of items need to be selected in order to maximise overall value, but is restricted to a weight. The classical GA chromosome for this problem will then have one position for each item, and the selected items will a value of 1. A simple one-point crossover is shown in Figure 2.3 (b) to illustrate how this operation works. After selecting two parents (in white and grey in the figure), one point is chosen to split the chromosome. Each part of a chromosome will then be combined with the other part of the other parent, generating two children which have half the information from each parent. In a mutation the genes are modified according to some rule. In Figure 2.3 (c) we show an example where three random bits are inverted. It is important to note that this example only shows a fixed length chromosome, which although common, might be insufficient for several problems. It is not uncommon for HH based on GA to use variable-length chromosomes.

The main difference for a GP is that it is used to evolve rules and pro-



(a) Example of GA chromosome.

(b) Example of GA crossover.

(c) Example of GA mutation.

Figure 2.3: Examples of representation and basic operations in a Genetic Algorithm.

grams rather than direct solutions. In a GP approach the usual representation is tree-based, where each leaf (or terminal) represents some variables, while the internal nodes (or functions) are the operators [109]. Figure 2.4 shows an example of a simple tree structure used in GP. This tree can be translated as $min((x + x), (x + (3 * y)))$, where $\{x, y, 3\}$ are the terminals and $\{+, *, min\}$ are the operators. Like the GA, they also go through the crossover and mutation processes as part of the evolution process, the difference being the type of operators used, which are usually specific for tree-based representations. For example, crossover and mutation operators are shown in Figure 2.5. The crossover transfers a sub-tree from one parent to the other, while the mutation adds a new sub-tree to the parent.

2.4.1 Evolutionary Hyper-Heuristics

Evolutionary Hyper-Heuristics are the ones which utilise EA methods as part of the Selection or Generation process. These methods provide some clear advantages when applied to each type. For example, for Selection HH they provide a clear way to improve the order in which operators are applied. This is because EA has an implicit online learning mechanism through its evolution process, carrying knowledge from the previ-

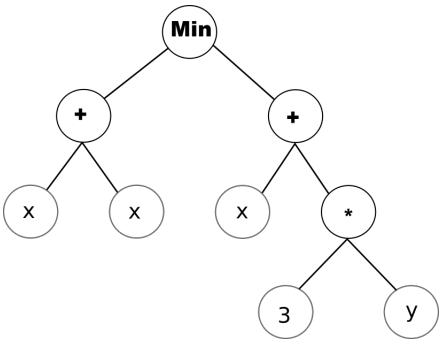


Figure 2.4: Representation of a Genetic Programming Tree.

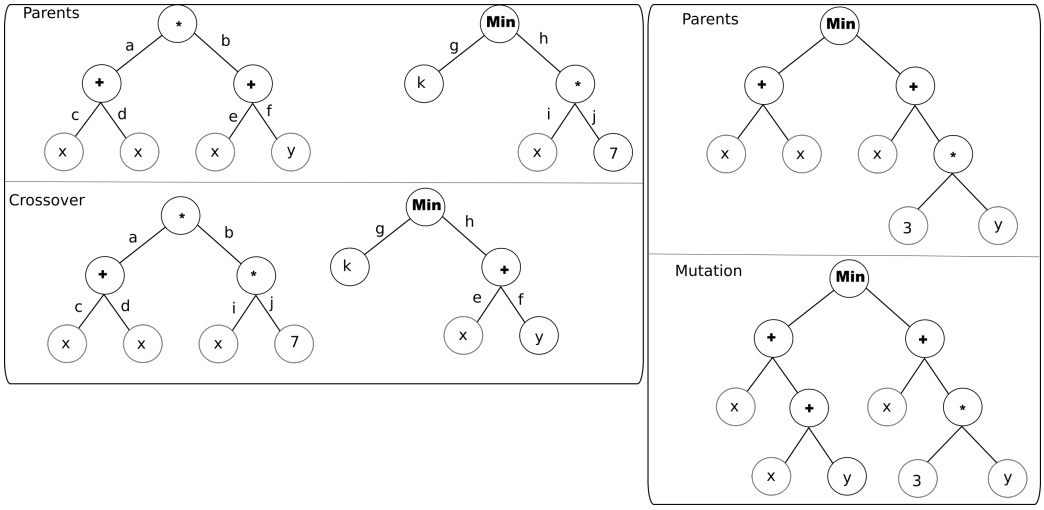


Figure 2.5: Examples of Genetic Programming Crossover and Mutation operators.

ous generation to the next. They can also be easily used for offline learning, where the evolution process is applied considering evaluating more instances during convergence.

GP is used as HH when the evolved trees are used to build a policy or heuristics. In [30], the authors explore the potential of GP as HH, applying it on SAT and bin packing problems. Several works have successfully applied GP for solving COPs, such as Job Shop Scheduling [23,201], Team Orienteering [129], Arc Routing [122] and VRP [97,172].

There are also GA-based Hyper-Heuristics. Usually used for selecting perturbation heuristics, the GA-based HH will evolve a sequence in which the operators (LLH) will be applied to an initial solution. It has been mostly applied in timetabling [148, 159], scheduling [85] and bin packing problems [149]. A recent review on EA as ML, including as HH, was presented in [5].

2.5 Related Work

In this section we talk specifically about the works developed for the VRP and LSVRP. As the VRP is not a new problem, the number of existing published work is over thousands[4], therefore we try to limit the scope of this survey to relevant work. We cover some of the baseline and state-of-the-art algorithms in both exact and non-exact approaches, as well as some of the manually and automatically designed methods that tackle the LSVRP specifically. We divide these methods by the type of approach and their size (VRP vs LSVRP).

2.5.1 Exact Methods for the VRP

Exact methods for the VRP often rely on branch-and-bound techniques (or similar, such as branch-and-cut, branch-and-price etc.). **Branch-and-bound** is an algorithm that intelligently searches partitions of the feasible solutions space into smaller subsets which can be solved more easily and producing better lower bounds [116]. The challenge is to find better branching points, or valid additional cuts that reduce the number of possible solutions. For example, one of the first branch-and-bound algorithms for the VRP was proposed in [35], where the authors add artificial depots to a TSP branch-and-bound approach. These artificial depots serves

[4] A *Google Scholar* search for the term "Vehicle Routing Problem" returns around 121000 results.

as splitting points for the TSP route, so each visit to them make a VRP route. They also propose using a minimal spanning tree to determine better bounds[5]. The number of artificial nodes, however, is a parameter N, and might be infeasible, which makes several runs necessary until a valid answer is reached. More recent branching-based approaches, such as the branch-cut-and-price by [144], will work similarly but adding very carefully designed valid cuts, requiring a vast knowledge in the mathematical background and techniques. Other notable branching-based methods are [13, 66, 96, 160].

Another technique is based on **Set partitioning** and **Column Generation** methods, where the problem is formulated as a set partitioning problem, considering routes rather than edges as decision variables. But since the number of possible routes is also very large, a column generation method is used to generate these routes iteratively. Examples of this approach can be found in [4, 14, 49].

The authors of [146] have proposed a generic solver for the VRP using a Column Generation and Branch-cut-and-price (BCP) method, finding better results than techniques created specifically for some of these variants, and is considered the state-of-the-art of the exact approaches.

Dynamic programming was also used for exactly solving the VRP, as in [36, 59], but have been since poorly explored for the classical variant.

A recent case is shown in [6], where a large-scale instance is broken into clusters to find and assign the number of vehicles, and a TSP is used to build the routes of the assigned vehicles. The gap to the optimal values, although better than some other exact methods, are still worse than those that heuristics find given the same time frame.

Although there is an active pursuit of more efficient exact methods, especially the branch-and-bound type of approach, they still fail to get competitive times when compared to non-exact approaches. Especially

[5] A few years later the work of [89] have shown that the shortest spanning 1-tree is a very good bound for the TSP.

for larger instances, where even days of execution time are required for some problems with less than 1000 customers. For example, in [144], an instance with only 320 customers took 39 days of execution time. Therefore, the study of heuristic methods are essential for finding feasible solutions within the minimal amount of time possible. More about exact methods can be found in [22, 110, 111, 152].

2.5.2 Heuristics for the VRP

The first heuristics go back to 1964 and were the method of choice for solving the VRP until the 1980s [186], and can be divided into the **constructive heuristics**, which greedily builds a solution step by step, and local improvement heuristics that improve an initial solution iteratively. Although these heuristics are not very efficient, they are essential to provide good initial solutions and improvement moves for more advanced methods. Here we present some of these heuristics from both types.

The first known heuristic for the VRP, the savings algorithm was proposed in [37]. The savings algorithm is based on a simple idea of connecting pairs of routes. The algorithm starts with trivial routes (n vehicles serving the n customers in back and forth routes), which are merged at each iteration maximising the loss of distance when connecting them, if feasible. Even though it does not have the best results, the algorithm is very fast and easy to implement, which is why to this day the savings algorithm is still used for creating decent initial solutions for other approaches.

Another constructive heuristic, the sweep algorithm [76, 195], builds the routes by considering an initial angle and "sweeping" a line segment across the customers starting a new route whenever the vehicle is full. This method attempts to put the customers that are close to each other (considering the angulation from the depot, forming sort of "petals") in the same route. The routes generated are usually not very effective depending on several features, such as the depot and customers locations, their

demands etc. But the method is also used as a starting solution which can be improved by other methods.

The Route-First Cluster-Second approach, from [139], builds a single TSP-like tour and then splits it according to the capacity of the vehicles. The Cluster-First Route-Second [62] creates clusters (groups) of customers and then find a TSP tour inside these clusters. Other proposed constructive heuristics can be found in [115].

The local improvement heuristics need an initial solution to improve on, which can be seen as a post-optimisation step [111]. This initial solution can be created using a constructive heuristics as the ones above, or can even be randomly created with little to no attention to feasibility. The improvements are based on the neighbourhoods chosen. For the VRP and variants, these neighbourhoods can be divided into intra-route and inter-route [111]. The intra-route neighbourhoods make improvements within each route as if a single TSP-tour, while inter-route moves happen to multiple routes simultaneously.

For intra-route neighbourhoods, the λ-opt is perhaps the most known, especially when $\lambda = 2$ or 2-opt (Two-Opt) [64], and the 3-opt ($\lambda = 3$) [45]. This heuristic, originally for the TSP, deletes edges in the tour to try to find a better way of reconnecting the nodes, and was generalised as λ-opt in [119]. However, it is known to be tricky to be efficiently implemented, requiring sophisticated data structures and programming techniques [112]. The parameter λ represents the number of deleted edges, and is also the complexity of the neighbourhood, i.e. $O(n^{\lambda})$ [90,186], which is the main reason the lower degree versions are more popular (2-opt and 3-opt). Finding the λ-optimal solution considering every customer will result in the optimal solution for that tour [90], when a TSP-tour has n cities if no improvement can be made considering the n-opt neighbourhood, then the solution is optimal. However, the complexity of such a neighbourhood is so expensive ($O(n^n)$) that this is not a process which is pursued, with very rarely any method applying a value for $\lambda > 3$ [90].

There is a popular implementation of the λ-opt being used to solve TSP (and even VRP) in [90], called the LKH2 (later improved and expanded to the LKH3 version [91]), exploring up to 5-opt moves (or even 6-opt moves in some special cases).

The Or-exchange (or Or-opt) heuristic [141] is a special case of the λ-opt, which is a sub-set of the 3-opt with size $O(n^2)$, achieving that by relocating sequences of visits [186]. Other than edges exchanges, another common approach is based on chain exchanges, i.e. relocating a sub-set of consecutive costumers to another part of the route.

For Inter-route neighbourhoods, the shift (also called insert) neighbourhood [142] will move a customer from one route to another. A swap (or 1-interchange) will exchange customers from two different routes. The 2-opt* (Two-Opt-Star) [154], deletes two edges from two routes and reinserts them into the other, therefore they assimilate some part of each other's route, as illustrated in Figure 2.6. The Cross-Exchange [178] will exchange two sequences of customers between two routes, one can be empty. Therefore it is a generalisation of the shift, swap and 2-opt* algorithms [186]. The inverse version of Cross Exchange, I-Cross [24], will do the same but will invert the order of visits for one of sequences. These two neighbourhoods have a complexity of $O(n^4)$, but can be limited by the size of the sequences, resulting in $O(S^2n^2)$, where S is the maximum size of the sequence. The relocation chain [10] is a recently proposed inter-route neighbourhood based on ejection chains [77]. The idea consists of relocating one customer into another route, and from that route relocate another customer to some other route, and so on. But the method requires some additional pre-processing and searching procedures, as well as some pruning techniques to make its use feasible since this move neighbourhood grows exponentially with the number of route chains [10].

Another type of neighbourhood that can be considered a third way (against intra and inter routes) are the destroy-and-recreate heuristics (or ruin-and-recreate). They work by deleting edges or customers in several

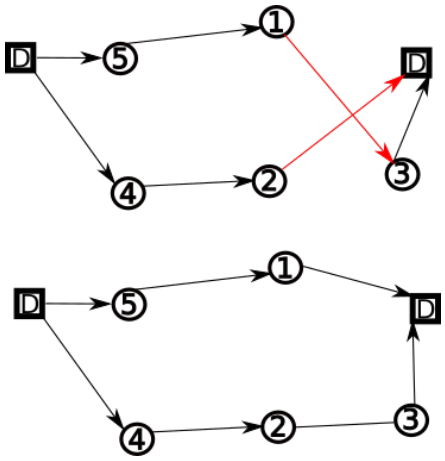


Figure 2.6: A simple example of the 2-Opt* operation.

points of the VRP solution, which are then reconnected in a different way, as in [171] and [170]. Somewhat similarly, the heuristics of [65] and [180] fix some customers, while the unfixed are re-assigned between the fixed ones. They do that re-assignment with an integer-programming model.

The complexity and size of these improvement heuristics, although seemingly small, are actually one of the biggest issues for larger scale problem. The number of necessary evaluations and solutions generated become the most time-consuming step of any method. Although not mentioned here, several other heuristics have been proposed for the VRP and variants. They usually use a combination of the neighbourhoods shown, apply different techniques, such as parallelism, or do a mix with different methods, especially exact methods for solving some sub-problem. More about heuristics for the VRP can be found in [25,39,67,103,111,115,179,186]

However, these heuristic methods are insufficient to find better solutions for more complex problems (other variants or different types of customer distribution), since they did not have mechanisms allowing escaping local optima [22,111]. In other words, these heuristics heavily rely on exploitation and are mostly greedy. But as shown in the next section, they became integral parts of most metaheuristics, guaranteeing themselves as

necessary components for solving the VRP.

2.5.3 Metaheuristics for the VRP

After the heuristics boom, metaheuristics became the leading methods for solving the VRP. They incorporate strategies that allow escaping local optima, finding a better balance between exploration and exploitation. We consider two types of metaheuristics here for the sake of organisation: individual-based (or neighbourhood-centred as put by [186]) and population-based. The first group focuses on iteratively applying improvement heuristics on an incumbent solution, while the second has several solutions which exchange information in some way to create new solutions. We also talk about some hybrid methods, which combine two or more algorithms in order to take the advantages of each one, and have achieved the best results overall.

For **individual-based** metaheuristics, several have been successfully applied for solving the VRP, we list some important work and briefly explain how they work. More about metaheuristics for the VRP and variants can be found in [39,70,74,153]

The **Simulated Annealing** (SA) [104] will allow a solution to deteriorate based on a temperature parameter. This method was inspired by the process of physical annealing, i.e., heating and cooling down a crystalline solid [140]. For optimisation problems, the algorithm allows worsening moves as part of the heating analogy (exploration phase), while improve the solution during the cooling down process (exploitation). The temperature parameter works as the probability of accepting worsening solutions, and it will decrease over the execution time (as if cooling down over time). Therefore, the algorithm starts with a focus on exploration, and turns to exploitation over time. For the VRP, the SA starts with an initial solution, usually from a constructive heuristic, and improves on it with a set of neighbourhoods which are chosen randomly [74]. Some improvements

have been done for the SA in the VRP context based on Record-to-Record, such as in [56, 118], where they add a limit to how much worse a solution can be accepted in the heating process.

Tabu Search (TS) [79] is a metaheuristic approach which keeps a record of previous moves in a list called *tabu* (hence the name). This list will be used to avoid repeating the same move for a given amount of time, based on a parameter, allowing worsening moves if the other available moves are in the *tabu*-list [72]. The works of [69] and [40] have successfully applied the TS to solve the VRP and VRPTW, respectively.

The **Guided Local Search** (GLS) [189] alters the objective function in order to leave locally optimal solutions. It does that by penalising the function considering some features of the problem [190]. The challenge comes from how to select these features. The standard one is the longest edges. A very efficient approach was recently proposed for the LSVRP by [9], where the authors consider routes characteristics discussed in [11] as part of the penalisation function, obtaining state-of-the-art results.

A **Variable Neighbourhood Search** (VNS) [134] will iterate through three main steps: shaking, local search and move [86]. When shaking, a random move in the neighbourhood is selected. A neighbourhood is then selected (either randomly or deterministically) and searched. Finally, if an improvement is found the method will start repeating the neighbourhoods, otherwise it advances to the next unexplored one. For the VRP, it has found some work have found success when adding some additional perturbation mechanisms and TS-like memories, such as in [107] for the LSVRP, [63] for the OVRP[6], and [33] for the VRP.

The **Iterated Local Search** (ILS) [16] does not consider the full search space, but rather based on some underlying algorithm, usually a local search. The ILS will explore a neighbourhood until a local optimum is found, then change the neighbourhood through a perturbation phase [124].

[6]Open VRP, a variant where the vehicles do not need to return to the depot at the end of each route.

The perturbation usually employs some sort of history and try to move the solution to another part of the search space. The solution found can also be accepted or not based on some acceptance criteria. The ILS works similar to a Hyper-Heuristic, since it can be combined with other metaheuristics approaches in its core. For the VRP and also other variants, perhaps the most well-known and still considered a state-of-the-art approach is the work of [176]. Their hybrid ILS combines an exact Set Partitioning formulation as one of the underlying algorithm while dynamically controlling the size of the exact model to avoid losing performance with larger scales. The authors find good results, improving several best results at the time, for several variants. One of the downsides of this approach is the requirement of a good solver for the exact model to be efficient. The algorithm also scales badly for larger instances ($500+$ customers).

The **population-based** metaheuristics will usually have a hybrid phase with some local search to be more effective. We list some of these metaheuristics and how they are applied to solve the VRP.

The **Ant-Colony Optimisation** (ACO) algorithm [50,51] is based on the behaviour of some ants species, which lay down pheromone in their trail allowing other ants to communicate. The main steps of the ACO metaheuristic consists of a construction phase, where the m ants will build solutions, then an optional search phase where each ant will explore using local search, ending it with the pheromone parameter being updated according to each ant experience [52]. The pheromone parameter and the decay factor (how fast this pheromone dissipates) play a huge role in the effectiveness of the algorithm. For the VRP, several ACO applications have been proposed, such as [17,27,162,198], where each ant will represent a vehicle and their goal is to build a feasible route, hence most are constructive-based. The pheromone enters as historical data and local search is also incorporated to improve each route.

Originally just an intensification step, the **Path-Relinking** (PR) [78,164] is an evolutionary metaheuristic which use solution recombinations to

evolve their populations. PR explores trajectories between elite solutions through neighbourhood moves in an attempt to find better solutions when moving from one point to another. It was applied to the VRP in [92], where it was combined with a TS to better explore the path between the points.

The **Particle Swarm Optimisation** (PSO) is a metaheuristic where each individual in its population is a particle associated with an objective value and velocity [101, 153]. The velocity replaces crossover and mutation in classical GAs, as it modifies and interacts with different particles pushing them to new solutions [153]. For the VRP, the work of [120] applies a PSO algorithm where each particle is a matrix of probabilities for each element (customer/column) to be in the vehicles' route (row).

Perhaps the most well-known population-based algorithm, the **Genetic Algorithm** [93] is inspired in the natural evolution process and has been extensively used for solving VRP and its variants. The work of [155] was the first to successfully propose an effective GA for the VRP. The author proposed several key design choices which managed to result in a competitive Hybrid-GA (HGA) for the VRP. The main differences are the addition of an improvement procedure as the mutation step, as well as additional specialised algorithms for population management, chromosome translation and evaluation. In other words, when mixing local search methods and neighbourhood moves. More recently, another Hybrid GA [185], later generalised for other variants in [187], got state-of-the-art results and is still considered one of the most relevant works for solving VRP and variants. The UHGS (Unified Hybrid Genetic Search, as they call it) also adds more specialised algorithms regarding population management (similar to the work of [155]) and more efficient evaluation procedures. This work was later improved for the VRP and LSVRP in [184], we talk a bit more about it in the next section, as the modifications also focused large-scale.

In [34] the authors introduce a series of ruin and recreate moves by exploiting VRP characteristics, through the use of **Slack Induction by String**

Removal (SISR). It became one of the state-of-the-art metaheuristics since it was able to find very good results for several of the datasets in literature. The SISR also presents an easy-to-reproduce algorithm, making it popular fairly quickly.

The Google OR-Tools[7], is also a tool for VRP that became popular recently due to being open access and provided by Google. However, it often performs much worse than the state-of-the-art metaheuristics.

All state-of-the-art algorithms for the VRP such as the UHGS [187], the Hybrid-ILS [176], utilise a hybrid approach with local search mechanisms. This is because local search has shown to be most effective when searching for a good solution. However, when purely used, they lack the capability to escape local minima, whereas the metaheuristics have carefully elaborated plans for such case. Therefore, the most efficient algorithms are the ones that have selected the best pool of neighbourhoods for their local search, and also have chosen efficient exploration methods which allows escaping those minimal points. Additionally, as pointed out in [9], they show that both the UHGS and the Hybrid-ILS, although keeping a very good solution quality, take up to hours of execution time for solving instances of 600 or more customers

Therefore, the main issues with the metaheuristic approaches for the VRP are regarding the large number of parameters and design decisions needed for achieving these good results, on top of their lack of scalability, which is covered next.

2.5.4 Addressing the LSVRP Heuristically

Traditional metaheuristics for the VRP might not scale well for the LSVRP, specially considering most of them ignore the size of the neighbourhoods. Another common issue is memory management, which is mostly disregarded by the above algorithms for non-large-scale problems. Most meth-

[7] Available in `https://developers.google.com/optimization/`

ods consider a distance table which is often calculated before execution, size increases can make this table unfeasible for several computer systems, with tables of up to $1.5GB$ for a 20.000 customer instance (according to [9]). Since this is only one of the data structures utilised in such heuristics, it can easily become a problem if unchecked. We now list some relevant work that solves the LSVRP and briefly report how they deal with the large scale.

The work of [158] proposes a constructive heuristic method for a real-world application for pick-up of employees in specific points. The heuristic is based in finding the outermost points and add them first, making their way through closer points, respecting capacity. Since it is constructive, the algorithm has no scalability issues regarding neighbourhoods. Although simplistic, the application of this method reduced both the total distance by around 10% and the number of utilised vehicles, and is one of the first work aiming a large-instance.

For metaheuristic development, in [118] the authors apply a SA for solving instances of up to 1200 customers. Some of these instances are proposed in the paper. Their SA limits the number of neighbours to be explored to the 40 nearest. The results are within 3% of the best-known solution at the time, and is achieved very quickly, averaging less than 4 minutes of execution time.

A two-phase method is proposed in [131]. The first phase focuses on creating an initial solution based on the cheapest reinsertion with a composite local search. The second phase is a GLS penalising longest edges, storing the values to avoid recalculation. Additionally, they employ a ranking system for the customers closer to the penalised edge, considering only some closest customers (based on parameters). Then, a destroy and recreate heuristic is applied to improve the solution, also considering a limited number of closest customers. Although they find very good results, even improving some best known solutions, the execution times are inconsistent and are still somewhat slow, taking up to 11 hours.

In a similar approach, the Guided VNS (GVNS) of [107] has achieved good CPU time and results. The algorithm also has two phases, where the first one is a VNS-like approach, building a solution starting from the farthest customers, adding more customers and applying neighbourhood moves. The second phase is a VNS which penalises the longest edges of two given route pairs, followed by inter-route moves. Then, the intra-route operators are applied if any modification happened.

A Quantum Annealing (QA) approach is proposed in [177]. QA is a metaheuristic based on the Path-Integral Monte Carlo method. The method works somewhat similarly to a SA, but with different energy equations and representations. Although they find competitive results, the method takes too long for finding such solutions.

A parallel heuristic method is presented in [182]. The approach consists of splitting the search space into smaller cells that can be solved with a parallel local search based on a ILS framework. The ILS is additionally applied in the border regions (where cells meet) trying to find better improvements since different routes are likely close in those areas. The authors do some analysis on how to split the regions, finding that some types of division are more effective in certain scenarios. They were able to solve real-world instances with up to 20000 customers within 30 minutes.

The work of [121] applies two distinct operators to determine whether a pair of routes are likely to have improving exchanges. Their Variable Neighbour Descent (VND) method explores the full neighbourhood, applying the inter and intra-route moves in pairs of routes if they have a distance or difference in angle between their centres of gravity smaller than a given threshold parameter. As no comparison to other methods is given, they only measure the effect of this threshold and how it improves execution time, without much loss of quality.

The work of [202] has shown an hierarchical decomposition method, based on an ILS framework. The LSVRP instance is divided into subgroups, finding a permutation of tasks and then splitting them to find a

solution, which is solved recursively. Then the solution is given to the adapted ILS procedure. The result is improved with a local search which are then fed to a new hierarchical decomposition scheme. The results, however, are not competitive, not even beating the traditional Savings algorithm for some instances.

An evolutionary multi-objective approach based on route grouping was proposed in [196]. The authors considered a divide-and-conquer strategy to do the decomposition based on three distinct objectives: *intragroup* distance, *intergroup* distance and intergroup balance in size. These objectives are considered to find better clusters which are then optimised with a Tabu Search-based local search. They show that their multi-objective approach outperforms another considering the K-means to create the groups. Overall, the method shows competitive results with execution times of less than 1 hour for instances of up to 1200 customers.

In [157] the authors propose a POPMUSIC[8] algorithm, which utilises the already mentioned BCP from [146] as a heuristic, hence it is a matheuristic. However, the method showed to be highly dependent on very good initial solutions and is seen more as way to improve already reasonable good solutions. Although it is able to solve large-scale problems, it does so by taking a long time (several hours). It was, however, able to improve some of the best-known-solutions from some of the benchmark instances.

Although these methods contribute to the literature of the LSVRP, some get more attention and importance due to their outstanding contributions, we list those next, starting with the one we use as baseline for most of our contributions in this thesis, *the KGLS*.

The Knowledge-Guided Local Search and other State-of-the-Art

The Knowledge-Guided Local Search (KGLS) [10] is a deterministic algorithm that was introduced by adapting an existing classical heuristic, into

[8]Partial optimization metaheuristic under special intensification conditions.

a powerful VRP solver. It was later expanded to deal with large-scale problems, in [9], solving instances of size up to 30000 customers (these come from a new dataset also introduced in [9]). The main differences between the standard GLS and the KGLS are in the carefully elaborated neighbourhood pruning[9], powerful local search moves (including a simplified version of the Lin-Kernighan heuristic [90,119], the Cross-Exchange [178] and the new Relocate-Chain) and the use of the width as part of the penalisation criteria. The width of a route was found to be a common characteristic of the better solutions in [11], and it can be defined by Equation 2.8. Hence, minimising the width could lead to better solutions.

$$width(i,j) = max(d_{iE}, d_{jE}) - min(d_{iE}, d_{jE}), E = Line(D, G) \tag{2.8}$$

The width is the distance to a line (E) that is traced from the depot (D) to the centre of gravity of a route (G), where d_{iE} and d_{jE} are the distances to E. More details on the width are presented on the original work [11].

The KGLS method works as follows: from an initial solution (they use a modified version of the Savings heuristic [37]) the solution is improved until a local optimum is reached. From this, there is a perturbation phase, where the solution is guided according to three different penalisation functions which are applied after each iteration (the badness functions, presented in Equations 2.9, 2.10 and 2.11). Then, another optimisation is applied, repeating these two phases until the stopping criteria. Equation 2.12 is utilised as the new objective to be optimised during the penalisation phase, which considers the penalised edges (with L being a parameter based on the Savings heuristics). Therefore, guiding the solution towards solutions that have a smaller width.

$$b^w(i,j) = \frac{w(i,j)}{1 + p(i,j)} \tag{2.9}$$

[9] Among the pruning techniques, a preliminary evaluation of a possible move which calculates the possible gains, avoiding the search if not positive. Another one saves only a pre-determined number of distances in memory, based on the closeness of the nodes.

$$b^c(i,j) = \frac{c(i,j)}{1+p(i,j)} \tag{2.10}$$

$$b^{w,c}(i,j) = \frac{w(i,j)+c(i,j)}{1+p(i,j)} \tag{2.11}$$

$$g(i,j) = c(i,j) + \lambda p(i,j) L \tag{2.12}$$

Here $p(i,j)$ represents the number of penalties the edge (i,j) already has. Then, the objective function (Equation 2.12) will indirectly remove the worse edges considering these three penalisation criteria alternatively and in this fixed order ($W \rightarrow C \rightarrow WC$). In function 2.12, λ represents the weight of the penalised edges (i,j), while L is the average cost of an edge on the initial solution.

More recently, [184] improved their original work (the already mentioned UHGS [187]) specifically to the CVRP (and LSVRP). In this new version, a new neighbourhood operator is introduced, the SWAP*. In it, there is an exchange of two customers between two distinct routes without an insertion in place.

Another method often ranked among the state-of-the-art is an extension of the already mentioned Lin-Kernighan-Helsgaun heuristic, from [91], the LKH3. It solves the VRP and LSVRP by transforming them into a symmetric TSP with some additional handling made for each specific variant of routing problems. Although it can be effective in solving large-scale problems, it is rather slow, as shown in [9], taking from days up to a month of execution time. Still, it is often used as a baseline for evaluating the quality of other LSVRP methods since it is able to solve the large instances without extra tuning.

2.5.5 Hyper-Heuristics for the VRP and LSVRP

Several HHs have been proposed to solve the VRP, across the different types of HH. As shown by [150], with the exception of the combination Generation HH and perturbation heuristic, all major types of HH have been used in some way to solve this problem, until the year 2018. The

used construction heuristics for the VRP are either the classic constructive heuristics such as the Savings (from [37]) used in Selection HH, or a set of functions and problem attributes for Generation HH [150]. The perturbation-based HH will use the neighbourhoods to modify a solution.

Although the Constructive Selection HH is mentioned in the VRP literature, it is never alone, always being accompanied by perturbation heuristics. This is because the traditional constructive heuristics employed, such as Sweep, Savings, Insertion [135] or other greedy heuristics, do not find competitive results and need to be further improved. For example, in [130] the authors utilise a greedy insertion algorithm and the sweep heuristic not only for generating an initial solution, but also as exploration operators. They apply six different neighbourhoods for exploitation.

Not named a Hyper-Heuristic, the work of [151] employs the same concepts of a Perturbation Selection HH. Their Adaptive Large Neighbourhood Search (ALNS) was designed to solve five VRP variants (VRP, VRPTW, MDVRP, S-DVRP[10] and the OVRP) which are converted to another more general problem to be solved by their framework, the Rich Pick-up and Delivery Problem with Time Windows (or RPDPTW). After conversion they solve the RPDTW through a series of neighbourhood moves which are selected by the proposed new adaptive layer to the LNS of [171]. This layer works similar to other HHs, operating at a higher-level when choosing which type of perturbation heuristics to apply in this unified type of problem. Although effective, converting these problem variants is not so straightforward, requiring additional design effort.

The work of [172] combines both Selective and Generation HHs in a two-stage algorithm. The first stage will build a heuristic using a Genetic Programming (GP) approach. GP can be viewed as a type of Generation HH since its tree-based structure results in a program rather than a solution itself. In their work, the GP builds a constructive heuristic using a set

[10]Site-Dependent VRP, a variant where a sub-set of the customers can only be served by some specific vehicles.

of several functions and nodes. The resulting constructive heuristic is then used to create the set of initial solutions. The second stage uses a perturbation Selective HH to improve the solutions from the previous stage, utilising a set of local search heuristics and crossover operators as the LLHs, selected at random. Although an improvement is shown regarding the initial solutions created by the resulting method, the overall improvement after the two-stages is not competitive with other metaheuristics.

In [55], the authors use a Generation HH method using a grammar-based form of GP, known as Grammatical Evolution (GE). GE uses strings expressed in *Backus Naur Form* which are translated into the resulting program. The pre-defined grammar translates these strings, having several steps such as finding an initial solution, ruining and recreating rules, and terminal points and functions where the routes components and functions are stored, respectively. The resulting heuristic is used to feed a VNS system to evolve a solution. In other words, the method creates a constructive heuristic and neighbourhoods which are used in a traditional VNS algorithm. The work, however, explores very little on the results, not finding competitive results even for small scale instances (less than 40 customers).

Firstly in a short paper [128] and later expanded in [127], the authors of this work propose another GE method for the VRP where the resulting string also contains the calls for improving the solution (perturbation heuristics). One of the possible improving moves is a deterministic variation of the ILS, basically a full metaheuristic. The grammar consists of four main elements which are applied consecutively. First a strategy is defined to create an initial solution (or partial solution). Then, constructive operators are used to build a feasible solution. The third phase will improve the solution. And, finally, a fourth phase where the process is repeated by a number of cycles to find a final and improved solution. The authors also do some tests for online and offline learning, however not much details are given, other than training with different instances for the offline mode. In the short version they present few good results for instances with up to

80 customers. On the new version, they found the best results for most of the 40 instances of up to 101 customers, even improving some of them.

Not many HHs have specifically dealt with the LSVRP. We could only find the paper of [167], which does tackle a large scale variant, but for the VRPTW. Nevertheless, we comment on it since the approach is relevant for this study. The authors apply a two-stage Math-Hyper-Heuristic to solve the problem and to deal with the large scale. The first stage divides the instance into m sub-parts which are fed into a column generation method to find an initial solution. The second stage is the Hyper-Heuristic phase, where a Perturbation Selection HH is applied to improve the solution given in the previous stage. The heuristics are selected using a Multi-armed bandit mechanism (MAB) which rewards each LLH according to its performance over time. Additionally, after the MAB selects the next heuristic, a Monte-Carlo approach is utilised to accept the resulting solution based on a probability, if it is a worsening move, since improving moves are always accepted. They utilise a set of 7 different LLH, finding competitive results against state-of-the-art algorithms. However, the impact of each stage is not clear. Column generation might be too slow for larger instances (they limit the tests on instances from 200 to 400 customers), and determining m experimentally might be unsuitable for other types of instances. No attention to scalability is given to the HH phase.

2.6 Summary

In this chapter we presented an overall idea of the current state-of-the-art for the LSVRP, introducing some basic concepts of the field, HHs, as well as an overview of classical heuristics and metaheuristics for solving the VRP. Most approaches apply, at least on some level, a manual expert-driven design decision. Although these decisions can perform well in several scenarios, there is no guarantee that they are generalisable. Not only these manual decisions might not work well on different kinds of in-

stances, they are also loosing on potential effectiveness, as they opt for parameter values or elements that are good on average, but might not cover well some cases.

The HH approaches that aim to minimise these downsides, however, are far from solving these issues. Not only that, but most of them fail to deal with the scalability of instances. In summary, the limitations that can be found in the LSVRP literature are:

- The metaheuristics and most common methods applied are tailored for specific benchmark instances that might not be generalisable for other applications, customer distribution or demands.
- Most methods heavily depend on manually designed pruning. This is done by experts and might not be the most suitable approach for different sizes or instance configurations.
- There are a limited number of HHs considering large scale problems. Most HHs do not present specific concerns regarding the scale of the problem. Therefore, a lot of possible approaches can be considered and tackled to find results that could not only be used across different datasets, but also lead to discoveries in understanding VRP solutions more closely. This is one of the motivations that [165] listed as why to study HHs: to understand more about the problem.
- The HHs also fail to provide any competitive advantage. Although several authors seem to agree that HHs should exchange quality to generality, theoretically they can be competitive as the manual designed decisions are one of the possible configurations of the heuristic search space. Hence, there is room to improve the quality of the HHs while also keeping their ability to be general, adapting or learning how to solve the problem in a better way.

This thesis aims to investigate and develop new methods to address these limitations.

Chapter 3

Learning Effective Initialisation

In this chapter we tackle the use of learning techniques to find effective initial solutions.

3.1 Introduction

Examining the search space of a complex problem such as the VRP is a challenging task. Some recent work attempts to understand the landscape of such a space [3], but still not much information can be extracted. If such a landscape could be fully known *a priori*, we could utilise that information to search for solutions directly in the most promising regions. As that is not the case, we still need to start the search from some point. There seems to be a correlation between the starting point and the quality of the local optima [143], therefore, where the search starts will have a high significance for the methods' performance. Hence, predicting the algorithm which performs best in such a task could be seen as a meta-learning problem [173]. However, determining the initial point is only the first step in the process, which will be improved afterwards.

Several VRP methods choose a random initialisation approach, especially the population-based ones, spreading their chances of landing in good regions of the search space. However, the LSVRP methods are still

prone to the quality of a single initial search space, since these commonly use a local search framework with one single solution. Therefore, it is crucial for such a point to provide some advantage.

However, this starting point was not heavily considered in recent literature, as it is thought to provide little gain when compared to high-performing improvement techniques. Hence, what ends up happening is that a generic construction method will be used based on just its overall quality or the designer's affinity to that heuristic.

We consider this an oversight. If a better initial region can be found, the search method would prove even more effective[1]. Machine Learning (ML) can help automatically identify those regions that have potential to include better local optima — and this could be achieved with some training data, not requiring expensive landscape analysis methods. For example, considering each constructive heuristic builds a solution in a certain way, leading to a specific region of the solution search space, the ML techniques could learn when to select each method to land on more promising regions. The approaches developed in this chapter attempts to learn where to land on the solution search space.

3.1.1 Chapter Goals

The key goal of this chapter is *to learn how to effectively use the initial solution to improve the search efficiency*. We introduce an analysis of the effects of the initial solution and evaluate them with ML techniques. We then use some of the solutions features to build novel GPHH terminals and a fitness function which are used to construct new initial solutions for a guided local search framework. This chapter has the following main objectives:

1. Provide an analysis of the impact of common initialisation heuristics to the improvement process.

[1]This argument could be valid for population-based methods as well, but it is not in the scope of our thesis to examine such cases.

2. Utilise different instance and solution characteristics as features in a learning model to predict which initialisation heuristic will perform better for a specific instance.

3. Introduce a new fitness function and terminals which are used in a GPHH approach to build solutions in specific regions of the search space that are easier to optimise.

3.1.2 Chapter Organisation

This chapter is organised as follows: Section 3.2 presents an analysis of the initial solution impact. Section 3.3 introduces the learning strategy applied to predict the best initialisation heuristic. In Section 3.4 we introduce our GPHH approach which builds a new initialisation heuristic. Finally, an overall Chapter summary is given in Section 3.5.

3.2 The Initial Solution Impact

There is little evidence of how much the starting point affects the performance of local search-based methods in the recent literature for the VRP. Most of these local search-based methods utilise a simple constructive heuristic to build the initial solution (as shown in Chapter 2), since they are mostly very fast (most implementations can find full solutions in less than one second, even for large-scale instances), and focus on the improvement phase which is more expensive (minutes to hours of execution). In this section we discuss this and present some experiments on the impact of the initial search and argue their relevance.

3.2.1 The KGLS and the Initialisation Heuristics

The Knowledge-Guided Local Search (detailed in Chapter 2) was chosen for most of the experiments in this thesis for two main reasons. First, the

KGLS is a framework fully built on a local search. While other state-of-the-art methods, like [184], use local search in their search process, they also perform different types of exploration unrelated to the local search, like crossover or the use of exact approaches (like in [176]). The KGLS, on the other hand, has all its steps involving the local search process, which matches our goals of optimising the local search configuration.

The second reason is the scalability. The KGLS was shown to perform efficiently even when the problem size increases, solving instances of up to 30000 customers within a very reasonable time (a couple of hours), having almost the same quality as LKH [91] that requires days for execution. Although this thesis focuses on the KGLS, the principles applied throughout the thesis can also be applied to other local search methods, including those within the other state-of-the-art metaheuristics, such as the local search steps used in [176, 184].

The KGLS originally uses a modified version of the savings heuristic introduced in [37]. Clarke and Wright's savings heuristic (CW) builds the solution by starting with trivial routes (n vehicles serving the n customers in back-and-forth routes), they are then merged at each following iteration maximising the loss of overall distance when connecting them, if feasible, based on a savings table (hence it is popularly known as the savings heuristic). The modified version introduced in [9] works about the same way, but it is optimised for large instances, where the savings table only holds the 100 closest customers (hence we use the acronym CW100), to avoid a very large (memory-wise) table to be recorded. Although this modification limits the search space, it was shown to be almost as good as the original CW heuristic for a lot of instances. We confirm this similarity in our results, but also argue that the original CW might still be a valid heuristic to be considered even in large-scale instances, as memory limitations tend to be fairly easy to overcome by technological advances.

Apart from these two methods, we also utilise other two popular approaches that are used in VRP as constructive heuristics: the Nearest Neigh-

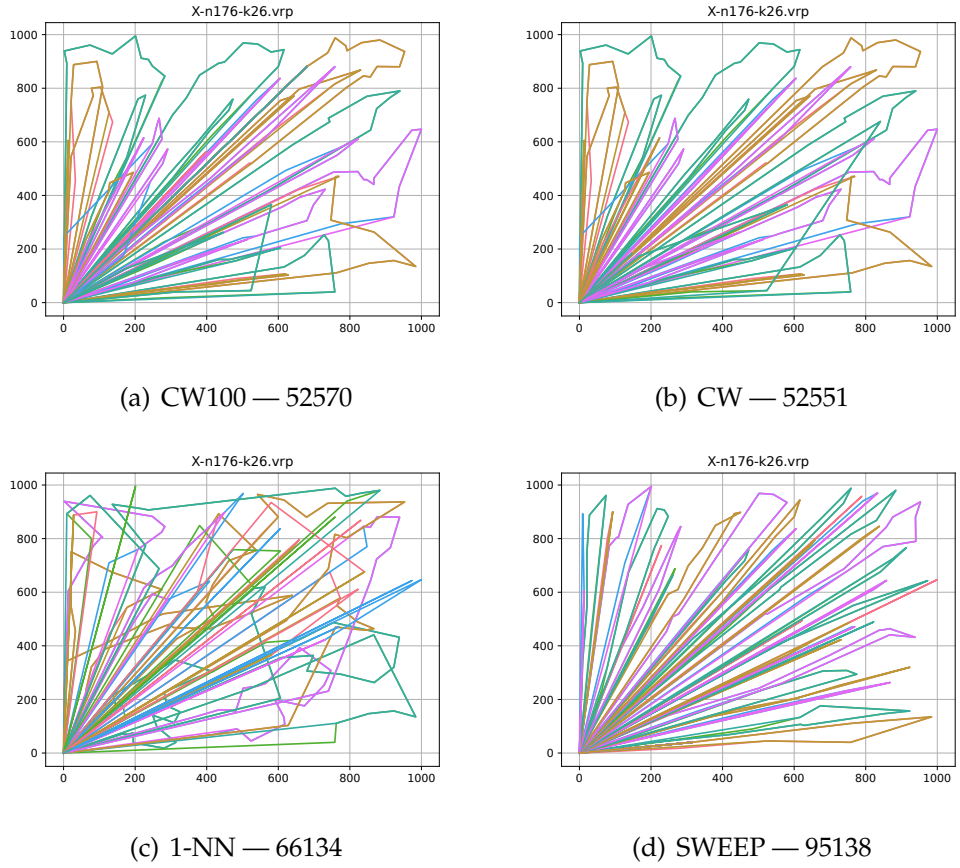
bour (1-NN) and the Sweep heuristic (SWEEP). The first was originally introduced in [18] for the Travelling Salesman Problem but was then adapted for the VRP. The idea is simple, starting from the depot, select the nearest customer and add to the first's vehicle route. Next, add the closest customer from the first customer and repeat the process until the vehicle's capacity is reached, skipping the closest customers which do not allow for a feasible addition (whose demand added would exceed the vehicle's total capacity load). Then, the next vehicle is chosen while selecting the closest unvisited customer from the depot, repeating the process as the previous vehicle. This greedy heuristic tends to perform very badly when compared to other constructive heuristics since it does a poor job of accounting for the next routes.

The Sweep heuristic [76, 195] builds the routes by considering an initial angle and "sweeping" a line segment across the customers starting a new route whenever the vehicle is full. This method attempts to put the customers that are close to each other (considering the angulation from the depot, forming sort of "petals") in the same route. The routes generated are usually not very effective, as the method also suffers from the greediness of maximising the vehicles' load. An example of the solutions generated by the four constructive heuristics for the *X-n176-k26* instance and their total costs are shown in Figure 3.1. We can clearly see that there is a big difference in cost and route structure (apart from the two savings methods, CW and CW100, which are fairly similar to each other), which illustrates how far these solutions are from each other.

3.2.2 Analysing the Initialisation Heuristics Performance

For the analysis, we first ran the above four constructive heuristics over the 100 instances of the "X" VRP dataset [183][2]. This dataset provides in-

[2]The instances' name represented as "X-ni-kj" are a code for their number of nodes (i) and the minimum number of vehicles (j).



(a) CW100 — 52570

(b) CW — 52551

(c) 1-NN — 66134

(d) SWEEP — 95138

Figure 3.1: Example of routes created by the 4 heuristics considered — for instance X-n176-k26: (a) CW100 (with cost 52570) (b) CW (with cost 52551) (c) 1-NN (with cost 66134) (d) Sweep (with cost 95138)

stances with different customer distributions (network topologies), such as Clustered (customers can be easily grouped), Random (customers are spread randomly) and the mix Random-Clustered (RC, where both clusters and random customers are present). The results presented here were obtained by changing only the algorithm utilised in the construction of the initial solution.

As can be seen in Table 3.1, it is clear that both CW and CW100 perform much better than the other 2 heuristics (CW wins overall by a bit, which

Table 3.1: The initial costs given by the different initialisation heuristics for some representative instances. The average represents the value over the 100 instances of [183].

Instance	Topology	CW100	1-NN	CW	SWEEP
X-n125-k30	C	**59659**	69054	**59659**	73647
X-n157-k13	C	**17831**	18979	**17831**	26418
X-n172-k51	RC	**48228**	64976	**48228**	69382
X-n233-k16	RC	**20433**	30950	**20433**	42959
X-n247-k47	C	**40870**	52358	**40870**	61692
X-n367-k17	C	25348	28746	**25343**	64245
X-n384-k52	R	**69524**	74913	69526	128751
X-n420-k130	RC	**112317**	140018	112604	147419
X-n469-k138	R	235279	242364	**234913**	330625
X-n524-k137	R	166509	217803	**165536**	240520
X-n641-k35	RC	68128	71182	**67988**	214738
X-n670-k126	R	158564	206695	**158545**	270490
X-n749-k98	C	**79313**	105117	79698	158132
X-n837-k142	RC	201348	207337	**201119**	355154
X-n979-k58	C	123528	143373	**123224**	328968
Average	—	66395.86	76384.24	**66352.22**	129836.1

is expected as it does not limit the savings table). Although Table 3.1 is showing only a few randomly selected representative (topology-wise) instances, the averages presented are for all 100 instances from the dataset, which are not fully presented due to space limitations. There is also no clear pattern on which topology is preferred for each instance. Based on this table, one might expect the same distribution for the solutions' quality after the local search phase, as both CWs are leading the "race" for a good final solution by quite a big margin. These results would match the findings of [15], which shows that some greedy heuristics can be impractical as starting solution for some problems.

However, as Table 3.2 shows, that is not true. The table presents a summary of the instances from the "X" dataset [183] (the same instances from

Table 3.1 are shown, with averages from all 100 instances as well), after 10 runs[3] of 15 minutes of the KGLS algorithm with each initial heuristic. A fixed time approach was selected rather than a number of fixed iterations as the first are dependent on the instance being solved[4], while time is a criterion that is employed in real-world applications.

Surprisingly, the 1-NN heuristic actually has a better overall final value over the 100 instances. Considering it is much worse in the initial stage at around 15% higher cost, the starting point was able to overcome the large gap and provide a final better solution on 34 instances (52 if including the ones where it matches the CW100). Perhaps even more surprising is the Sweep heuristic, which starts at an enormous 95% GAP to the CW100 and still was able to finish by a mere 0.04% higher cost in the same time frame. The topology also does not seem to have any influence on this, just like in the initial solution. All constructive methods were able to perform the best for at least 13% of the 100 instances considered. These additional statistics are condensed in Table 3.3.

Table 3.3 also shows why just choosing a fixed option such as the Nearest Neighbour (or any other in particular) is a sub-optimal strategy. The table shows that although overall the performance can be better (improvement of up to 0.11%), it would also hinder a lot of solutions (39 out of 100 which would have an average worsening of 0.15%, if considering the 1-NN). This agrees with the No-Free Lunch theorem [194] since it is clear that some instances are better solved by different methods, but also per-

[3] As the KGLS is deterministic, unlike the original GLS, and so are the selected heuristics, there is no need for statistical analysis on these runs. We still run it 10 times to account for possible oscillations in computer performance.

[4] It would not be fair to give a fixed number, e.g. 5000 iterations, for a 100 customers instance and the same number to a 1000 one, as the first would obviously benefit more than the second. Although time can be argued the same, some instances have a more complex search space. In those cases, a fixed number of iterations could be very expensive. For example, the larger instances can take a couple of seconds per iteration (multiplying that by a big number can easily lead to hours of execution).

Table 3.2: The results obtained by KGLS with different initial solutions on some representative instances. The average represents the value over the 100 instances of [183].

Instance	Top.	CW100	1-NN	CW	SWEEP	Best
X-n125-k30	C	56167.6	56144.5	56169.5	**56129.3**	-0.07%
X-n157-k13	C	**16876**	**16876**	**16876**	**16876**	0.00%
X-n172-k51	RC	45664.0	45789.8	45664.0	**45643.0**	-0.05%
X-n233-k16	RC	19360.0	19363.6	19360.0	**19359.9**	0.00%
X-n247-k47	C	37701.0	**37662.0**	37685.0	37701.6	-0.10%
X-n367-k17	C	23442.8	**22898.3**	23415.5	23033.0	-2.32%
X-n384-k52	R	66585.0	66599.0	**66475.0**	66490.0	-0.17%
X-n420-k130	RC	108374.5	**108370.9**	108425.0	108379.3	0.00%
X-n469-k138	R	223403.0	224029.8	**223263.0**	224154.3	-0.06%
X-n524-k137	R	157292.0	**156045.5**	157716.2	157529.0	-0.79%
X-n641-k35	RC	**64175.0**	64302.5	64223.0	64444.0	0.00%
X-n670-k126	R	152500.0	**149960.8**	152206.0	152102.7	-1.67%
X-n749-k98	C	78365.0	78344.5	78273.0	**78258.0**	-0.14%
X-n837-k142	RC	195403.0	**195258.7**	195393.0	195419.0	-0.07%
X-n979-k58	C	120061.0	119889.8	**119794.0**	119855.0	-0.22%
Average	—	63699.1	**63630.0**	63688.1	63721.5	-0.16%

form worse for others.

Table 3.3: A summary of additional statistics for the preliminary experiments over the 100 instances of [183].

	CW100	1-NN	CW	SWEEP
Total Best	28	**34**	16	22
Avg. GAP to CW100	—	**-0.11%**	-0.02%	0.04%
No. of instances worse performing than CW100	—	48	**38**	57
Avg. GAP when worse	—	**0.13%**	0.16%	0.17%

One might argue that it is much easier to find improvement in a worse solution than for an already good one. Although that is true, it would not

explain how the worse solution ended up being better. A counterargument for that could be: an already good solution leads to a local optimum much faster, and escaping it could be harder for the selected local search, hence the initialisation methods that produce worse initial solutions are less bound to such traps. However, if that is the case, and given the complexity of the search space of a problem such as the VRP, it would be also equally easy to trap the worse initial solutions in their own bad local optima, not allowing for a better solution to be found given the same operators. In fact, given that this experiment is bound in time (all heuristics run for 15 minutes), we can argue that the time lost to make the obvious improvements on the worse initial solutions would make up for the best initial solutions escaping their local optimum. Therefore, there should be other factors contributing to such differences.

This analysis shows that (1) a better initial solution does not necessarily lead to better final solution after the search; (2) no initialisation method is always the best. This leads us to believe that there are some characteristics in each initial set of routes that allow for the local search method to find an "easier" path to a better final solution. If such characteristics exist, they could be extracted and, therefore, a machine learning method could be used to learn their patterns. Doing so would allow for the local search methods to be tuned for the given instance based on the extracted characteristics of these easy (and fast) to compute heuristic methods. Although the gains are potentially small, learning which initial method to be used can provide a gain that is essentially free, as there is no need to modify the underlying complex local search method. This concept could theoretically be applied to systems already deployed with minimal effort (if the initial solution process is detached from the main core of the search).

3.3 Learning to Select Initialisation Method

Since different initialisation heuristics can perform the best on different instances/scenarios, a natural research question is whether we can use machine learning to automatically select the best initialisation heuristic for a given instance. In this section, we will investigate this research question.

The idea is that there are features in the solutions that can give a hint on how the search space "looks like" to the search method, and we could use those features to do such prediction. However, it would be a challenging task to define those features. For that we turn to literature, where several features and how to extract them have already been proposed. As shown in Chapter 2.5.4, the features proposed by [11] were applied successfully to distinguish good and near-optimal solutions, so they were the primary candidates for our approach. At first glance, some features would also make sense for this task, such as the number of vehicles used and the total solution cost for each method. Therefore, we also added those two additional features. A brief description of the features is presented in Table 3.4, where we divide them based on whether they are specific to the solutions built by the heuristics, or to the instance-specific ones, which are used as a form of normalisation, since they remain the same between different heuristics. For more details on the other features the reader can follow [11].

To learn the relationship between these features and the best initialisation heuristic we need a classifier. Since it is not in the scope of this thesis to find the most suitable classification algorithm for this task, we utilise a diverse set of ML techniques, which are described in Section 3.3.3.

3.3.1 Labelling the Data

Having the features and the ML methods, we still need one final element for training them: the labelled data. Although there might not be a ground-truth solution since that would depend on several factors such

Table 3.4: Solution-specific ($*$) and Instance-specific ($+$) features. All features are proposed in [11], except for Cost and NTS.

Feature	Description
$*$ Cost	Total cost
$*$ NTS	No. of vehicles/trucks used
$*$ S1	Avg. number of intersections per customer
$*$ S2	Longest distance between two connected customers per route
$*$ S3	Average distance between depot to directly-connected customers
$*$ S4	Average distance between routes centres of gravity
$*$ S5	Average width per route
$*$ S6	Average span in radian per route
$*$ S7	Average compactness per route, measured by width
$*$ S8	Average compactness per route, measured by radian
$*$ S9	Average depth per route
$*$ S10	Standard deviation of the number of customers per route
$+$ I1	Number of customers
$+$ I2	Minimum number of routes
$+$ I3	Degree of capacity utilisation
$+$ I4	Average distance between each pair of customers
$+$ I5	Standard deviation of the pairwise distance between customers
$+$ I6	Average distance from customers to the depot
$+$ I7	Standard deviation of the distance from customers to the depot
$+$ I8	Standard deviation of the radians of customers towards the depot

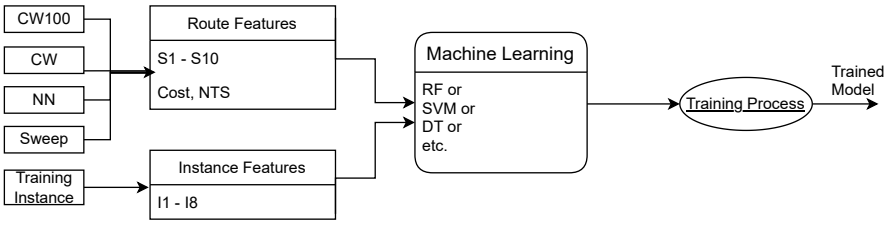
as the operators used and the time given to run them, we still believe their difference in performance (as shown in Section 3.2) is not a coincidence, but rather their characteristics that lead to specific regions of the solution search space. Therefore, we utilise the difference in performance of the KGLS in a 15 minutes run as the label for each class, where the label is the best performing initialisation heuristic. For example, the instance "X-n469-k138" has the best performing solution starting with the CW heuristic, hence its class would be labelled as CW. We do the same labelling for all instances, and, in case of ties (such as instance "X-n157-k13", which all heuristics find the best solution) the instance gets labelled with the default

method (CW100). Hence, the number of instances in each class is the same as presented in Table 3.3.

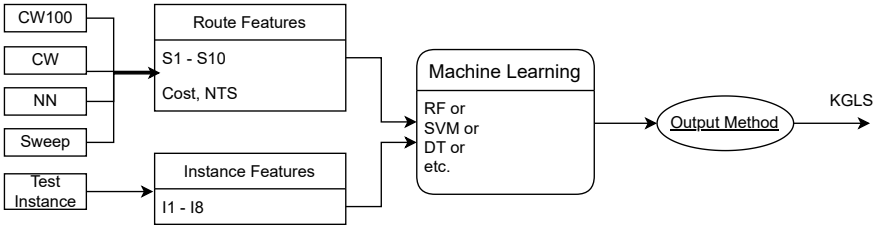
3.3.2 Machine Learning Approach

Given the labelled data, we can now train our ML methods and learn how to select the initialisation heuristic. The training and test processes are shown in Figure 3.2. For each training instance, first, we extract the instance-specific features. Then, we apply the 4 methods (CW100, CW, 1-NN and SWEEP) to the instance to generate the corresponding solutions and extract the solution-specific features. These are fed into the ML method of choice which will predict the output considering the labelled instance class as the expected output. Then, when applicable (as it depends on the ML method), the feedback process happens. After the training, we obtain the best classifier for selecting the best initialisation heuristic. Then we move on to the test phase. During the test phase, the 4 methods are first applied to the test instance. From each of the 4 solutions, we extract the solution-specific features, which are used in the ML prediction step. The ML output will be matched to the generated label (as specified above) and we can calculate the accuracy.

In the case of deployment of the method for a real operation, which would include unseen and unlabelled data, the ML would still output a predicted initialisation heuristic, which would then be used by the KGLS as its starting solution, moving on to the improvement phase. It is important to note that this process is feasible due to the heuristics used being incredibly fast, allowing us to collect all these features within the first few seconds of execution, and not damaging the overall performance of the algorithm across most scenarios (which we assume would require the algorithm to run for more than a couple of minutes).



(a) Training Phase



(b) Testing Phase

Figure 3.2: An overall flowchart for the: (a) Training phase (b) Test phase

3.3.3 Experiment Design

The experiments were done using the ”X” dataset presented in [183]. Apart from the different topologies as already mentioned in Section 3.2, this dataset has instances with distinct: depot locations (either at a Random location, at the Centre or at a Corner), customer demand distribution (either the same demands or varying them, and also with different ranges), number of vehicles (from small to large) and number of customers (ranging from 100 to 1000). This dataset provides sufficient diversity of features and topologies for our methodologies.

The ”X” dataset is randomly split into $70:30$ for training and testing, respectively. Additionally, we perform the training and test on 30 independent random data splits, allowing us to measure the robustness of the features across distinct data splits. As mentioned in Section 3.3, all the 100 instances were labelled by running the KGLS for 15 minutes and selecting the best-performing initialisation heuristic.

The utilised ML methods are the following:

- Random Forest (RF)
- Linear Support Vector Machines (SVM)
- Decision Tree (DT)
- Multi-layer Perceptron (MLP)
- K-Nearest Neighbour (KNN)
- Gaussian Process (GPC)
- RBF Support Vector Machines (RBF)
- Ada Boost (ABC)
- Gaussian Bayes Network (GNB)
- Quadratic Discriminant Analysis (QDA)

These classification algorithms are taken with their default implementations and default parameters from the Scikit python library [145]. These methods were selected because they are diverse enough to validate that the learning is not by random chance. The use of some of these methods could also help identify some feature relevance. For example, RF has a built-in feature importance method in the Scikit library, allowing us to evaluate which features are more relevant. We do not enter the merit of explaining these methods, not only because of it being out of scope for this thesis, but due to the complexity required to do so.

3.3.4 Results and Discussions

Now we explore the results of the ML test, comparing them to the use of a single initialisation heuristic. It is important to note that the accuracy of the results is not of the type ”correct-or-wrong”. This is because selecting

the wrong class does not mean the solution will be worse due to the existence of ties. For example, in an instance where CW100 and CW have the same final solution, its label would be CW100, but selecting CW would provide the same solution quality.

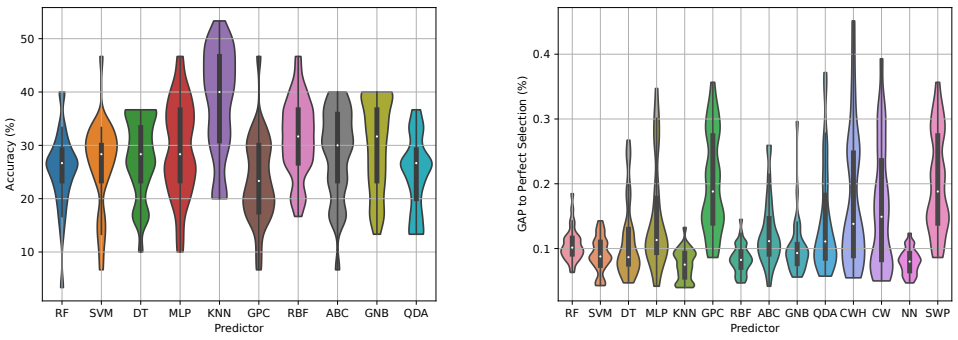
As can be seen in Table 3.5, which shows one single test run (out of the 30) to exemplify the data, the accuracy for some methods can be quite low. However, when looking at the overall costs, most ML methods were able to learn how to select initialisation heuristics with some degree of success. When comparing to the baseline (i.e. when utilising the default approach of only using one fixed initialisation heuristic, the CW100), most of the 10 methods were able to learn how to improve it, with the exception of GPC, for several of the runs. We also compare these predictors to the optimal selection, i.e. if all best heuristics were to be selected perfectly. Several methods were just barely worse, even with very low accuracy, indicating that the learning strategy was able to succeed.

Table 3.5: Results for all the considered ML methods and their respective costs for one of the runs with 30 randomly selected test instances. They are also compared to the default mode (using only CW100) and to a **theoretical** 100% accuracy method.

Method	**CW100**	**Best possible**	**RF**	**DT**	**SVM**	**KNN**
Accuracy	—	100%	43.33%	43.33%	33.33%	**60.00%**
Average Cost	85466.8	85192.8	85430.2	85446.5	85230.1	**85228.2**
Difference to Default	—	-0.3206%	-0.0428%	-0.0237%	-0.2769%	**-0.2792%**
Difference to best possible	0.322%	—	0.279%	0.298%	0.044%	**0.041%**
Method	**MLP**	**GPC**	**RBF**	**ABC**	**GNB**	**QDA**
Accuracy	33.33%	16.66%	53.33%	46.66%	26.66%	46.66%
Average Cost	85236.2	85448.9	85234.1	85243.3	85458.7	85240.5
Difference to Default	-0.2699%	-0.0210%	-0.2723%	-0.2616%	-0.0095%	-0.2648%
Difference to best possible	0.051%	0.301%	0.048%	0.059%	0.312%	0.056%

To verify that these values are not random, Figures 3.3 summarise the performance of the chosen methods over the 30 runs. The first violin-plot (Figure 3.3 (a)) presents the accuracy, which varies quite a lot, but also

shows that KNN seems to perform the best, followed by the RBF-SVM and MLP. Both linear-SVM and GPC also are able to find good peaks, but fail to generalise well, having a very large variance, although this can be argued for most methods.



(a) Accuracy (b) Gap to optimal classification

Figure 3.3: The overall performance of the ML methods over the 30 runs by two different statistics: (a) Accuracy (b) GAP in solution cost to optimal classification.

When looking at their ability to select good heuristics (i.e. not the best, but still better or equal to the baseline), we see a slightly different trend. The second violin plot (Figure 3.3 (b)) presents this information, with the GAP to the best possible selection. Here we also add the baseline performance (as CWH) — which shows how the default strategy would miss on quality over the different splits of the data, and the additional initialisation heuristics (CW, 1-NN and SWP), in order to verify whether it is worth to select a fixed mode. In the figure, we can see that a lot of the methods are consistently better than the baseline. In this graph we see less variability to the results, meaning the methods are likely miss-classifying instances with either one of the same quality or with the second-best heuristic. Here we notice that KNN also performs the best, but this time followed by the linear-SVM as well as the RBF. Although a few outliers exist, these meth-

ods still have a higher chance of finding a heuristic selection that improves over the baseline. A notorious exception, when compared to fixed strategies, can be seen in the Nearest Neighbour. The 1-NN shows a good performance overall and with less variance than other heuristics and is arguably better than half the methods.

3.3.5 Further Tests and Feature Analysis

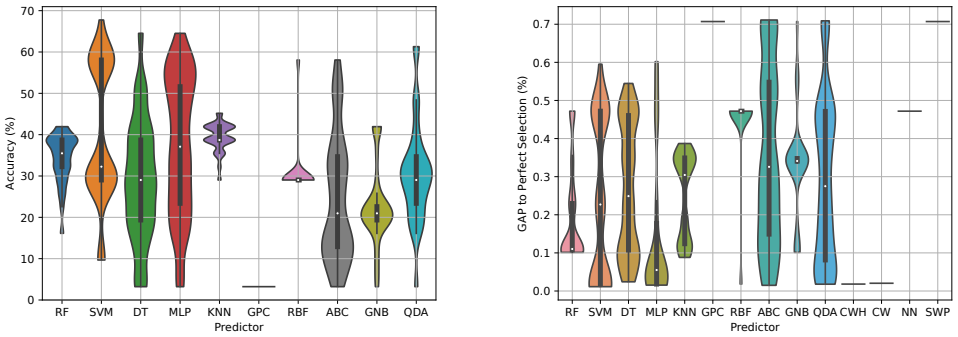
To verify the generality of the applied learning, since we have a limited number of instances to train, we also test the ML methods in both Li [118] and Golden [80] datasets. As they have very different characteristics from the "X" VRP dataset, we can validate if the features used are enough to differentiate the initialisation heuristics performance. These datasets vary by the number of customers from 240 up to 1200, although having a smaller number of instances (12 for the Li dataset and 20 for the Golden one). In Table 3.6, we show the number of instances in which each initialisation heuristic performs the best. We test with all the 32 instances after each training split from the previous experiment.

Table 3.6: Number of instances that each initialisation heuristic outperforms the others, Li [118] and Golden [80] datasets.

	CW100	CW	1-NN	SWEEP
Li	9	1	2	0
Golden	9	2	7	2

As for testing the algorithms in this new data, we show the results in Figures 3.4 (a) and (b). Accuracy-wise, we see a different behaviour, where most methods have a higher classification accuracy. The most notable difference is the drop in performance from the KNN method, while the MLP and SVM actually improve their performance. Performance-wise, we see a different picture from the first test. Here, most methods struggle to outperform the baseline. However, it is important to note that the lack of

instances and how closely their solutions are, play a big role in this performance. For example, the difference between the best-performing mode, the baseline CW100, and the second-best, 1-NN, is only 0.002%, indicating that they are almost the same in the 10 minute run. These results, however, do not discourage our findings, as they still show some insights into initialisation heuristics behaviour.



(a) Accuracy

(b) Gap to optimal classification

Figure 3.4: The overall accuracy and performance of the ML methods over the 30 runs considering the Li and Golden datasets.

We also look into the features for more clarification on the ML behaviour. For this, we look into Random Forest's ability to rank the importance of the features. We show the top 15 features in Figure 3.5 for the RF's best run. We can see that the solution features are playing a bigger role in the classification (which is expected from the showings in [11]), and varying between the heuristics, which is expected as a way to differentiate them. As can be seen, the two features introduced (cost and NTS) also seem to be relevant in distinguishing the methods' performance. These features make sense as they are often quite different between methods (as clearly seen in Figure 3.1). The only instance characteristic that showed in the top 15 was the average distance to the depot (feature i6), which can indicate the density of the instance can be decisive for normalising the so-

lution's characteristics. One thing is important to note, however, is that the importance of these features is never of high value — meaning they are used to differentiate the classes but not strong enough to make a clear distinction.

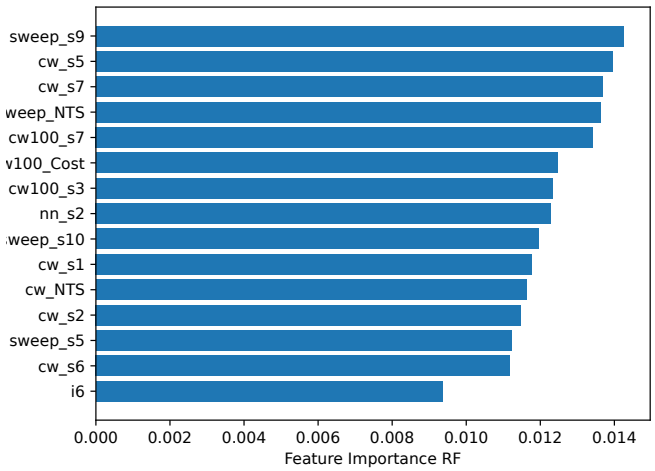


Figure 3.5: Feature ranking from one of the runs of the Random Forest.

3.3.6 Summary

In this section we introduced the use of different ML techniques to predict the best performing initialisation heuristic. We utilise several features from the considered constructive heuristics and feed them to our training model — in addition to the data that we labelled with some preliminary experiments. We were able to achieve good classification accuracy and improve over the default implementation.

3.4 GPHH for Effective Initialisation

In the previous Section we developed a learning strategy to predict the best initialisation heuristic. In this section we present the idea for using one of the most relevant solution characteristics to build a new solution. Our proposed Genetic Programming Hyper-Heuristic is mainly based on evolving a rule which is used to build a LSVRP solution in a more promising region of the search space. This is done by considering the width of a route, which was shown to be one of the main common metrics to the closeness to optimal solutions, as shown in [11].

3.4.1 Novel Terminals using the Width

The proposed GPHH utilises the standard tree-based representation, as most GPHH do (such as in [8,97,172,193,201]). The GPHH utilises a terminal and a function set to build the routing policy given a fitness function, which undergoes an evolutionary process to learn how to do so. The novelty of this algorithm takes place on the new terminal set and the fitness function. For the terminal, 3 new ones are proposed, as follows:

- WINACR - The width of the current position after adding the node considered. This terminal calculates the width of the current route when adding a new customer. Equation 3.1 is used, by considering the last customer from a route (i) and the candidate customer (j).
- WICR - The average width of the route being considered. This terminal calculates the average width of the current route evaluated. Equation 3.2 is used for the route considered.
- AVGWI - The average width of all routes in the solution. Calculated with Equation 3.3.

The equations that are used to calculate the new features are:

$$width(i,j) = max(d_{iL}, d_{jL}) - min(d_{iL}, d_{jL}) \tag{3.1}$$

$$RouteWidth(r) = \frac{\sum_{(i,j)\in r} width(i,j)}{||R||} \tag{3.2}$$

$$AvgWidth(s) = \frac{\sum_{(i,j)\in E} width(i,j)}{||E||} \tag{3.3}$$

Equation 3.1 calculates the width of a single edge between customers i and j and the line $L = Line(D, G)$, for more details please refer to Chapter 2. Equation 3.2 calculates the width of the route r by calculating the width of all sequential edges (using Equation 3.1) and dividing by the number of nodes in the route. It is important to note that the two edges from the depot are utilised in this calculation and, hence, $||R||$ is the number of edges in the route minus one (or number of nodes including the depot once). Finally, Equation 3.3 calculates the average width for solution s. It does so by calculating the widths of all edges used in the solution divided by the number of edges.

3.4.2 Incorporating Width to the Fitness Function

The second novelty for this approach, the fitness function (as can be seen in Equation 3.4), also takes into account the width metric.

$$N_p^t = \frac{\sum_{(j,k)\in E} d_{jk}x_{jk}}{BKS^t} + AvgWidth(t) \tag{3.4}$$

The function determines the fitness value N by normalising its cost to the Best-Known-Solution (BKS) for instance t and invidual p. The same is not done to the average width (Equation 3.3) because there is no standard value for the width — some longer width solution can also have a lower cost. This implies a greater weight towards the width, which is the new component that we want to exploit. The rest of the equation is detailed as follows: d_{jk} is the distance between nodes j and k, and the variable x_{jk} is a binary value which is 1 if there exists a connection between j and k in the current solution, and 0 otherwise.

We do this to mimic the behaviour of the penalisation step of the KGLS. Although that step is used to escape local optima (hence making the solution slightly worse), it also guides the solution to a place where it finds the next local optimum (often better than the original one — although not necessarily the case). An example of this can be seen in Figure 3.6, where a best-so-far versus current solution is plotted for two instances. In the figure we clearly see that some steps of the penalisation phase driving the solution upwards, these points are worse but can be closer to new better local optimum. As finding these very good local optima is very hard, we introduce the fitness function with the width expecting it to build a solution in one of the peaks near them. These runs were gathered with our own implementation of a GLS with the same penalisation functions as the [10], which we will call KGLS*.

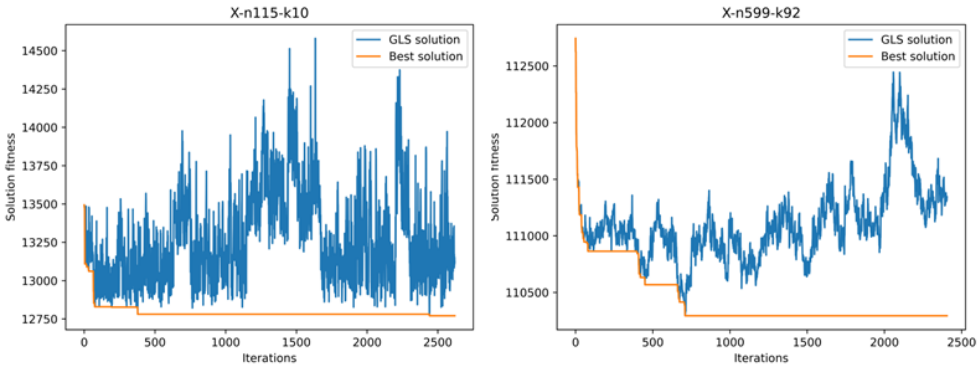


Figure 3.6: Examples of how the penalisation criteria guides the solution into new local optima.

The KGLS*, however, is an expensive algorithm. To evaluate the effectiveness of the new terminals and fitness function we would need to run the GLS for each individual tree's initial solution of each generation of our evolutionary process. On top of that, the instances considered have a wide range of cost values which makes it more difficult for the training process to evaluate the effectiveness of a given individual. We opted utilising only

the proposed fitness function to evaluate the individuals.

As to avoid bias towards instances that have a larger width — as these can vary a lot by instance — and also to speed-up evaluation, we apply the function to all training instances during one evaluation. As one instance is evaluated with Equation 3.4, the individual (p) of the population will have its fitness value by considering the sum for all instances, which is shown in Equation 3.5, representing the fitness for one individual. How to build the routes that provide these fitnesses is presented next.

$$F_p = \sum_{t \in Instances} N_p^t \tag{3.5}$$

3.4.3 Route Construction

The route construction algorithm is used to build the solution considering the current GP tree and instance. As shown in Algorithm 3.1, the procedure builds the routes simultaneously, unlike the standard approach which builds them in sequence (i.e. one route after the other). This way we expect to build a solution in a more balanced way between routes, as this algorithm considers the global scope rather than a local one. The GP-tree will be translated into a rule which is used to rank the routes in which the customer can be added. This translation is done by calculating the terminals values for each route and the customer being added. All customers are evaluated by their index sequence and the target route is selected according to the GP-tree. The algorithm tries to add all customers with a minimal number of routes. The initial number of routes is an approximation considering the capacity and total demand, calculated by dividing the total demand of the instance by the vehicle's capacity. The customers are ranked one at a time for each route. The highest-ranked route will have the customer added there.

Algorithm 3.1 Route construction method.

```
1: procedure ROUTE CONSTRUCTION(GP Tree T, Instance I)
2:     Find initial solution to i according to F's Initialisation part heuristic
3:     Number of routes = ⌈TotalDemand / VehicleCapacity⌉
4:     for each Customer c ∈ I do
5:         for each Vehicle v do
6:             Apply function tree T(c, v)
7:         Calculate and Select highest ranked vehicle
8:         if No vehicle is feasible then
9:             Add new vehicle
10:            Select new vehicle
11:        S = Add customer c to selected vehicle's route last position
12:    return Built solution S
```

3.4.4 Genetic Operators

Algorithm 3.2 Genetic Programming training method.

```
1: procedure GP_TRAINING(Training Set T)
2:     Find initial solution to i according to F's Initialisation part heuristic
3:     Initialise Population at random
4:     while Number of Generations not reached do
5:         for each Individual i ∈ Population do
6:             for each Instance t ∈ T do
7:                 Evaluate i with given t by Equation 3.4
8:             Compute fitness of i by Equation 3.5
9:         Select Parents
10:        Crossover
11:        Mutation
12:    return Best individual S
```

As the focus of this algorithm is more related to validate the new terminals and fitness function, the genetic operators are chosen from traditional methods, with a limited depth for avoiding very large trees. The overall algorithm for the GP training can be seen in Algorithm 3.2. First, the population is created at random. Then, the population is evaluated ac-

cording to the methods mentioned beforehand. The selection method is a Tournament Selection. The best individual is always kept in the population, unchanged. The crossover is a one-point crossover, where sub-trees are exchanged from the selected parents. For mutation, a new random sub-tree is added in the place of one of the leaves. Both crossover and mutation operators do not overflow the maximum depth of the tree.

3.4.5 Experiment Design

For the GPHH experiments, the "X" dataset is also used and we split it into training and test sets. But here the training set is fixed and includes all the instances with less than 200 customers (total of 21 training instances), while the test set includes all the remaining instances (79 instances). We introduced three new terminals for our GPHH, here we also list the set of all terminal nodes used, shown in Table 3.7. The other terminals are the traditional ones already utilised in the construction of VRP solutions.

Table 3.7: Set of terminals for the GPHH

Acronym	**Description**
DFD	Distance from considered customer to the depot
DEM	Demand of considered customer
DFCP	Distance from considered customer to the last customer added to the route
AVGD	Average distance from remaining non-served customers to the customer considered
AVGP	Average distance from remaining non-served customers to the last customer added to the route
CWD	Clarke and Wright [37] savings distance
MCDC	Most constrained demand
WINACR	Width of the current route after adding considered node
WICR	Width of the current route
AVGWI	Average width of all routes

To fully evaluate the performance of the proposed operators and that

they are indeed guiding into a better region of the search space, the experiments were designed in a way to validate this. The solution created with the proposed GPHH is expected to be improved quickly into a better final solution. To measure this, the solutions created with the proposed GPHH approach are improved by the KGLS* algorithm.

The KGLS* uses the parameters in Table 3.8 and utilised the following set of intra and inter (bold) route moves:

- Swap: Swaps the position of a pair of customers within the same route.
- Two Opt: Removes two edges and adds two new ones within the same route.
- Three Opt: Removes three edges and add three new ones within the same routes.
- **Cross Exchange**: Exchanges a pair of sub-routes of size k between two routes.
- **Swap Star**: Swaps the position of a pair of customers on different routes.
- **Relocate**: Relocates a customer to a different route.

Table 3.8: Knowledge-Guided Local Search parameters.

Parameter	**Value**
Execution time	10 min
Number of Penalised Moves (P)	10
Number of Closest Customers	30
lambda (λ)	0.1

Another GPHH (GPHH^2) was also trained to compare the effect of the new terminals. The GPHH^2 utilises the same function set (Table 3.7) and

fitness function (Equation 3.5) as the proposed GPHH. The only difference is that it is trained without any of the width-based terminals. Therefore, this GPHH^2 can measure whether these terminals are influential in finding a better solution. The compared GPHH methods use the parameters present in Table 3.9.

Table 3.9: Genetic Programming parameters.

Parameter	**Value**
Population size	1000
No. of Generations	50
Cross. Rate	0.8
Mut. Rate	0.2
Max depth	8
Tournament Size	7
Number of runs	15

Two main experiments were realized. The first experiment compares the proposed GPHH as the starting point versus the traditional Savings initialisation heuristic [37] (CW), also used the KGLS*. The best individual from the proposed GPHH will create the initial solution for the KGLS and measure how much the final solution was improved, in a fixed amount of time (10 minutes in these experiments). Additionally, the methods are compared on how fast do they improve the solution. The solution is also compared to the original KGLS implementation[5], also run for the 10 minutes.

The second experiment compares the proposed GPHH versus the GPHH^2, to validate the effectiveness of the newly designed width-related terminals in helping GP finding better initialisation heuristics.

[5]Available in the website: https://antor.uantwerpen.be/routingsolver/

3.4.6 Results and Discussions

The first results show the performance of the predicting model of our Machine Learning approach. Next, we present our GPHH results comparing the search performance from our initial solution to the original CW100. Then, we compare the performance of our method without the fitness function proposed and do a brief analysis of the top tree generated.

We compare the classification accuracy and the overall quality given by the different methods. We also present some analysis on additional instances and feature relevance for one of the approaches.

Impact of initial solution: To measure how much the initial solution influenced the evolution, the experiment was done to compare how the final solution was improved with the same amount of time. Table 3.10 compile these results in comparison to the Best-Known-Solution (BKS)[6].

Table 3.10: Results compared to the Best Known Solution (BKS). In bold show significance in the comparison between the KGLS* and our GPHH. Underlined are the overall best.

Instance	**KGLS to BKS**	**KGLS* to BKS**	**Proposed GPHH Avg. to BKS**	**GPHH Best to BKS**
X-n200-k36	0.29%	2.28%	1.70%± 0.63%	0.84%
X-n204-k19	0.52%	1.05%	1.16%± 0.19%	0.35%
X-n209-k16	0.25%	1.69%	1.68%± 0.29%	0.62%
X-n214-k11	0.67%	2.46%	3.78%± 1.70%	1.75%
X-n219-k73	0.07%	0.15%	0.15%± 0.02%	0.07%
X-n223-k34	0.59%	1.79%	1.55%± 0.18%	1.26%
X-n228-k23	0.37%	1.37%	1.90%± 0.65%	1.13%
X-n233-k16	0.54%	1.12%	1.47%± 0.33%	0.93%
X-n237-k14	0.24%	0.82%	1.29%± 0.55%	0.29%
X-n242-k48	0.47%	1.23%	1.34%± 0.15%	0.72%
X-n247-k47	1.11%	2.60%	2.06%± 0.36%	1.34%

[6]Values gathered from the dataset website: http://vrp.atd-lab.inf.puc-rio.br/index.php/en/

Table 3.10: Results compared to the Best Known Solution (BKS). In bold show significance in the comparison between the KGLS* and our GPHH. Underlined are the overall best (cont.).

Instance	**KGLS to BKS**	**KGLS* to BKS**	**Proposed GPHH Avg. to BKS**	**GPHH Best to BKS**
X-n251-k28	0.60%	1.64%	1.61%± 0.15%	1.01%
X-n256-k16	0.05%	0.80%	1.58%± 1.01%	0.75%
X-n261-k13	0.43%	2.00%	2.53%± 0.97%	1.41%
X-n266-k58	0.64%	1.31%	1.33%± 0.12%	0.91%
X-n270-k35	0.45%	0.95%	1.10%± 0.12%	0.95%
X-n275-k28	0.16%	0.97%	1.17%± 0.16%	0.81%
X-n280-k17	0.56%	2.28%	3.59%± 0.72%	2.96%
X-n284-k15	0.80%	2.31%	2.92%± 0.52%	2.25%
X-n289-k60	0.86%	1.56%	1.63%± 0.17%	1.16%
X-n294-k50	0.41%	1.65%	1.61%± 0.17%	0.97%
X-n298-k31	0.41%	1.34%	2.02%± 0.37%	1.55%
X-n303-k21	0.61%	1.72%	2.05%± 0.45%	2.06%
X-n308-k13	1.17%	2.39%	3.97%± 0.96%	1.64%
X-n313-k71	0.93%	1.49%	1.61%± 0.13%	1.19%
X-n317-k53	0.07%	0.41%	0.66%± 0.15%	0.28%
X-n322-k28	0.58%	1.94%	1.86%± 0.17%	1.53%
X-n327-k20	0.40%	1.33%	1.82%± 0.38%	1.43%
X-n331-k15	0.24%	1.28%	1.71%± 0.45%	1.14%
X-n336-k84	1.03%	1.66%	1.82%± 0.18%	1.42%
X-n344-k43	0.71%	1.57%	1.71%± 0.16%	1.43%
X-n351-k40	0.82%	2.58%	2.56%± 0.33%	1.52%
X-n359-k29	0.93%	1.52%	2.11%± 0.22%	1.42%
X-n367-k17	0.51%	2.04%	3.63%± 0.63%	1.43%
X-n376-k94	0.09%	0.32%	0.30%± 0.04%	0.24%
X-n384-k52	0.44%	1.39%	1.31%± 0.16%	0.89%
X-n393-k38	0.53%	1.93%	1.99%± 0.24%	1.42%
X-n401-k29	0.54%	1.05%	1.56%± 0.58%	0.81%
X-n411-k19	1.89%	2.70%	4.94%± 1.44%	4.81%
X-n420-k130	0.56%	1.35%	1.18%± 0.11%	0.71%
X-n429-k61	0.43%	1.58%	1.77%± 0.11%	1.54%

Table 3.10: Results compared to the Best Known Solution (BKS). In bold show significance in the comparison between the KGLS* and our GPHH. Underlined are the overall best (cont.).

Instance	**KGLS to BKS**	**KGLS* to BKS**	**Proposed GPHH Avg. to BKS**	**GPHH Best to BKS**
X-n439-k37	0.39%	0.93%	0.96%± 0.14%	0.71%
X-n449-k29	0.75%	2.55%	2.87%± 0.23%	2.44%
X-n459-k26	0.20%	1.80%	2.33%± 0.32%	1.84%
X-n469-k138	0.66%	1.20%	1.11%± 0.08%	1.06%
X-n480-k70	0.38%	1.27%	1.60%± 0.21%	1.28%
X-n491-k59	0.75%	1.78%	2.03%± 0.18%	1.90%
X-n502-k39	0.11%	0.57%	0.63%± 0.06%	0.58%
X-n513-k21	0.44%	2.16%	2.72%± 0.39%	2.05%
X-n524-k137	1.77%	2.66%	2.44%± 0.47%	1.82%
X-n536-k96	0.81%	1.55%	1.45%± 0.19%	1.22%
X-n548-k50	0.25%	1.00%	0.91%± 0.09%	0.83%
X-n561-k42	0.63%	1.71%	2.24%± 0.26%	1.75%
X-n573-k30	0.19%	1.39%	2.36%± 0.47%	2.04%
X-n586-k159	0.53%	1.19%	1.08%± 0.14%	0.94%
X-n599-k92	0.54%	1.22%	1.35%± 0.15%	1.30%
X-n613-k62	0.72%	2.27%	2.51%± 0.26%	2.33%
X-n627-k43	0.31%	1.78%	2.80%± 0.61%	1.99%
X-n641-k35	0.39%	1.90%	3.18%± 0.52%	2.40%
X-n655-k131	0.19%	0.49%	0.51%± 0.04%	0.45%
X-n670-k126	3.28%	5.33%	2.68%± 0.59%	1.94%
X-n685-k75	0.70%	1.72%	2.05%± 0.21%	1.62%
X-n701-k44	0.22%	1.55%	2.24%± 0.25%	1.73%
X-n716-k35	0.69%	2.60%	3.44%± 0.49%	3.48%
X-n733-k159	0.61%	1.39%	1.65%± 0.14%	1.43%
X-n749-k98	0.58%	1.69%	2.14%± 0.30%	1.75%
X-n766-k71	1.04%	2.84%	5.65%± 0.91%	4.34%
X-n783-k48	0.49%	1.60%	4.03%± 0.54%	3.37%
X-n801-k40	0.07%	0.96%	3.14%± 0.84%	1.62%
X-n819-k171	0.61%	1.16%	1.58%± 0.22%	1.37%
X-n837-k142	0.46%	1.26%	1.37%± 0.18%	1.15%

Table 3.10: Results compared to the Best Known Solution (BKS). In bold show significance in the comparison between the KGLS* and our GPHH. Underlined are the overall best (cont.).

Instance	**KGLS to BKS**	**KGLS* to BKS**	**Proposed GPHH Avg. to BKS**	**GPHH Best to BKS**
X-n856-k95	0.34%	0.83%	0.99%± 0.13%	0.73%
X-n876-k59	0.42%	1.67%	1.95%± 0.23%	1.86%
X-n895-k37	0.20%	2.54%	4.53%± 0.90%	3.26%
X-n916-k207	0.43%	0.91%	1.30%± 0.14%	1.15%
X-n936-k151	3.40%	6.22%	3.79%± 0.94%	2.97%
X-n957-k87	0.47%	0.84%	1.35%± 0.22%	1.05%
X-n979-k58	0.55%	1.23%	4.03%± 1.75%	1.92%
X-n1001-k43	0.40%	2.85%	4.06%± 0.50%	3.44%
Average	0.61%	1.54%	2.09%	1.64%

A statistical test (Wilcoxon Rank-Sum test with a significance value of 0.05) was performed to validate the proposed method's performance when compared to the KGLS* with the Savings as initialisation heuristic. Table 3.10 highlights the cases where there was an advantage for the GPHH. The values in boldface mean that there is a significant advantage over the proposed GPHH or the KGLS*. The best values found are also highlighted when the GPHH achieves that. In total, the proposed GPHH was able to have a significant advantage in 8 out of the 79 instances, and no advantage was found in other 23 instances. In other words, for about 40% of the instances the proposed method was just as good or better on the average case for all runs. Even though the initial solution has a much worse value when compared to the Savings heuristic, as can be seen in Figure 3.7. Additionally, when analysing the best-case performance, the proposed GPHH shows very promising results. In this case, the proposed starting solution was even able to beat the state-of-the-art for 4 different instances, using simpler heuristics. It also beats the KGLS* on average. Figure 3.7 also shows the performance of the best-case, which starts badly but outperforms the KGLS* in the end.

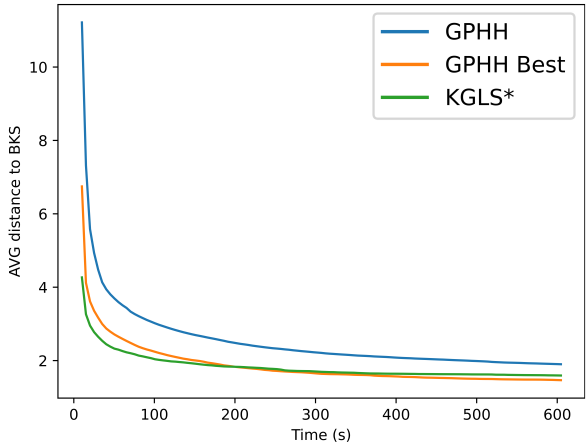


Figure 3.7: Comparison between the proposed GPHH average and best case to the KGLS* performance.

Impact of proposed terminals: This test measures if the GP novel terminals have any influence on the quality observed. Surprisingly, by comparing to the GPHH^2 (which only uses the objective function)the results show that the fitness function is the major contributor to the quality achieved. As can be seen in Table 3.11, the GPHH^2 outperforms the version with the terminals. This can be an indication that the extra terminals make the search more difficult by increasing the size of the search space. Whereas the fitness function is enough to guide the solution towards a region of the search space which can be quickly improved.

Table 3.11: Results compared to the Best Known Solution (BKS) for the two compared GPHHs.

Instance	**GPHH Avg. to BKS**	**GPHH^2 Avg. to BKS**	**Instance**	**GPHH Avg. to BKS**	**GPHH^2 Avg. to BKS**
X-n200-k36	1.70%	1.89%	X-n429-k61	1.77%	1.66%
X-n204-k19	1.16%	1.03%	X-n439-k37	0.96%	0.92%
X-n209-k16	1.68%	1.64%	X-n449-k29	2.87%	2.71%
X-n214-k11	3.78%	2.55%	X-n459-k26	2.33%	2.47%
X-n219-k73	0.15%	0.14%	X-n469-k138	1.11%	1.12%
X-n223-k34	1.55%	1.56%	X-n480-k70	1.60%	1.51%
X-n228-k23	1.90%	1.68%	X-n491-k59	2.03%	2.07%

Table 3.11: Results compared to the Best Known Solution (BKS) for the two compared GPHHs (cont.).

Instance	GPHH Avg. to BKS	GPHH^2 Avg. to BKS	Instance	GPHH Avg. to BKS	GPHH^2 Avg. to BKS
X-n233-k16	1.47%	1.33%	X-n502-k39	0.63%	0.58%
X-n237-k14	1.29%	1.24%	X-n513-k21	2.72%	2.36%
X-n242-k48	1.34%	1.35%	X-n524-k137	2.44%	2.55%
X-n247-k47	2.06%	2.49%	X-n536-k96	1.45%	1.45%
X-n251-k28	1.61%	1.50%	X-n548-k50	0.91%	0.88%
X-n256-k16	1.58%	1.34%	X-n561-k42	2.24%	2.15%
X-n261-k13	2.53%	2.49%	X-n573-k30	2.36%	1.78%
X-n266-k58	1.33%	1.29%	X-n586-k159	1.08%	1.11%
X-n270-k35	1.10%	1.00%	X-n599-k92	1.35%	1.47%
X-n275-k28	1.17%	1.01%	X-n613-k62	2.51%	2.39%
X-n280-k17	3.59%	2.57%	X-n627-k43	2.80%	2.11%
X-n284-k15	2.92%	2.43%	X-n641-k35	3.18%	2.60%
X-n289-k60	1.63%	1.67%	X-n655-k131	0.51%	0.52%
X-n294-k50	1.61%	1.65%	X-n670-k126	2.68%	2.60%
X-n298-k31	2.02%	2.17%	X-n685-k75	2.05%	2.10%
X-n303-k21	2.05%	2.16%	X-n701-k44	2.24%	1.95%
X-n308-k13	3.97%	4.18%	X-n716-k35	3.44%	2.80%
X-n313-k71	1.61%	1.66%	X-n733-k159	1.65%	1.58%
X-n317-k53	0.66%	0.50%	X-n749-k98	2.14%	2.08%
X-n322-k28	1.86%	1.74%	X-n766-k71	5.65%	4.42%
X-n327-k20	1.82%	1.90%	X-n783-k48	4.03%	3.59%
X-n331-k15	1.71%	1.54%	X-n801-k40	3.14%	2.10%
X-n336-k84	1.82%	1.82%	X-n819-k171	1.58%	1.54%
X-n344-k43	1.71%	1.72%	X-n837-k142	1.37%	1.41%
X-n351-k40	2.56%	2.38%	X-n856-k95	0.99%	1.07%
X-n359-k29	2.11%	1.98%	X-n876-k59	1.95%	1.79%
X-n367-k17	3.63%	3.20%	X-n895-k37	4.53%	3.62%
X-n376-k94	0.30%	0.29%	X-n916-k207	1.30%	1.35%
X-n384-k52	1.31%	1.41%	X-n936-k151	3.79%	3.28%
X-n393-k38	1.99%	2.01%	X-n957-k87	1.35%	1.15%
X-n401-k29	1.56%	2.05%	X-n979-k58	4.03%	2.44%

Table 3.11: Results compared to the Best Known Solution (BKS) for the two compared GPHHs (cont.).

Instance	GPHH Avg. to BKS	GPHH^2 Avg. to BKS	Instance	GPHH Avg. to BKS	GPHH^2 Avg. to BKS
X-n411-k19	4.94%	4.27%	X-n1001-k43	4.06%	3.57%
X-n420-k130	1.18%	1.14%	**Average**	2.09%	**1.91%**

In fact, when comparing to the KGLS*, the GPHH^2 converges much faster than the proposed GPHH with the terminals. As shown in Figure 3.8, the time required for the GPHH^2 initial solution to surpass the KGLS* is less than half of that of the proposed GPHH. Additionally, the improvement for the best case is on average 1.39% at the end of execution. Hence, the new fitness function plays a bigger role for the quality of the final solution presented. This can be linked to the selection pressure guiding the solutions towards the minimisation of the width.

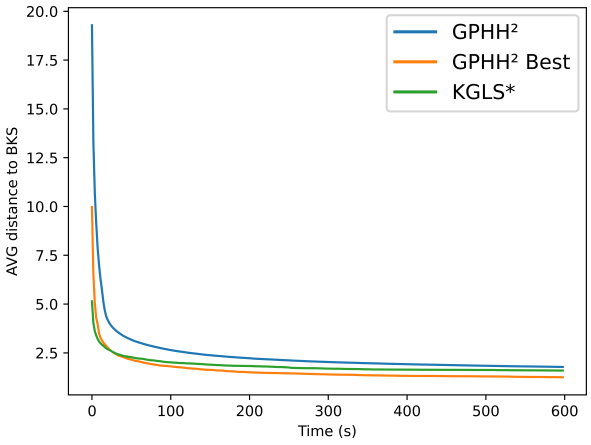


Figure 3.8: Comparison between the GPHH^2 average and best case to the KGLS* performance.

Analysing the GPHH with the novel terminals, looking at the tree generated by the best run, in Figure 3.9, we see that the terminals are used in several leaves. Although the repetitions might be happening due to the lack of bloat control in our algorithm — this can be argued because it was

not in our scope to increase the GP algorithm quality in itself — it also shows that the method selected these nodes without any pressure to do so, indicating their contribution to the method's performance.

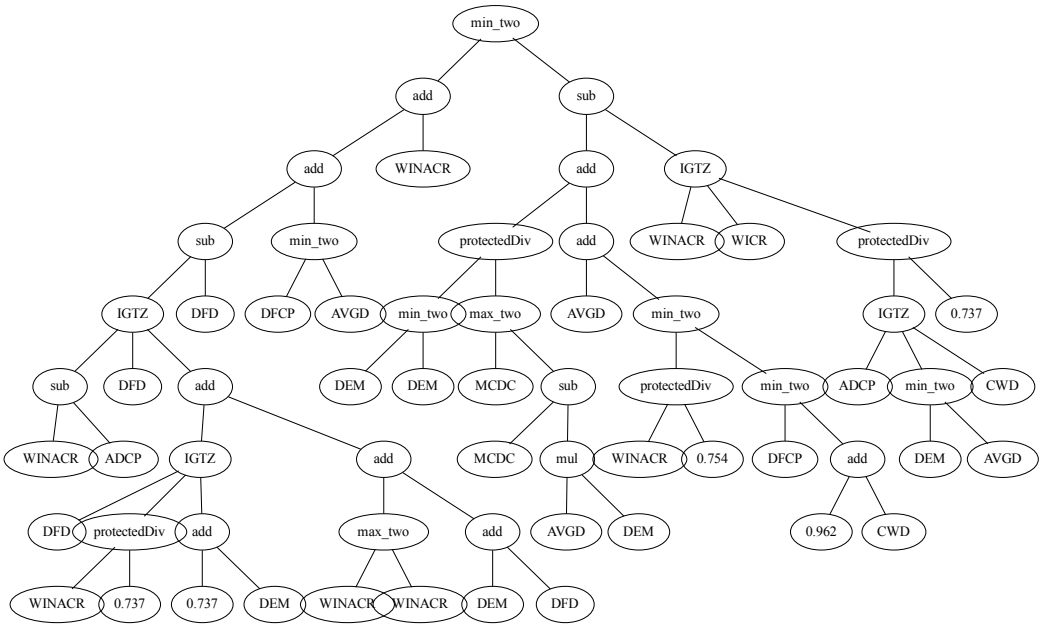


Figure 3.9: Evolved tree with the new operators.

3.4.7 Summary

This section presented our novel GPHH which aimed to create solutions that are not necessarily better, but are able to be improved faster than the traditional savings heuristic. We introduced some new terminals and fitness function to achieve this. The results, although underwhelming, were able to show some good best-case scenarios that motivate this research on being further developed.

3.5 Chapter Summary

In this chapter, the goal was to bring light to the use of the initial solution as a way to improve the local search performance — and that such goals were not necessarily achieved by finding better initial solutions. To do so, we introduce two new machine learning approaches to predict the best initialisation heuristic and to build a new easier-to-improve route.

We show an analysis of the impact of the initial solution performance with some preliminary experiments, measuring the difference in performance across four constructive heuristics — CW, CW100, 1-NN and Sweep. In order to take advantage of these findings, we proposed a classification approach that uses several solution and instance features to learn which of the traditional constructive heuristics would outperform in a given scenario. We also define a strategy to label the data and show the effectiveness of the approach on unseen instances. The results showed that the prediction was successful even when misclassifying the instances, shown by the difference in performance being small. This was achieved by randomly splitting the training data over 30 runs and showing that the overall costs are reduced. The KNN method seemed to outperform the others in both prediction accuracy and solution quality. An unexpected result comes from the outperformance of the Nearest Neighbour heuristic across most instances considered, a heuristic that is often used as a bad example of a greedy heuristic.

We also introduce a GPHH approach that is used to build a new solution without the focus on solution cost, but rather on a mix between cost and width, with the latter having a larger weight. We also achieve that by proposing new terminals, a new route construction method and a new fitness function. These components were able to work together and provide solutions that improve faster than the default heuristic, and in a few cases even outperforming the original KGLS with CW100. Although the new proposed terminals have shown to be less effective towards the goal

of improving the solutions, the overall results still show the effectiveness of the proposed GPHH successfully guiding towards a different region of the search space which is more promising. And when considering the time in which the solution outperforms the KGLS* (in the best-case), as shown in Figures 3.7 and 3.8, GPHH can also be seen as more efficient. Especially considering that this is achieved just by changing the starting solution.

3.5.1 Consideration on Scalability and Computational time

The two approaches introduced in this chapter scale well with the problem size. The first approach has a very small increase in computational time based on the instance size, almost negligible. This is because the ML method only considers the set of features as its input. Although some of these features are built using the initial solution, they are mostly calculated either by the number of customers or number of routes, which are very small. Of course, the following KGLS step with the selected initial heuristic will keep the same complexity, as that step is independent from the ML output. The second approach, will also only be slightly affected by the scale of the problem. The calculation of routes when evaluating the heuristics, as well as when applying the best heuristic to build the initial solution, can be calculated by iteration the number of customers. The tree evaluation might be a bit more expensive than the previous approach. This is because each customer must be evaluated individually.

Regarding computational time, a similar expectation can be found. The first approach will have insignificant impact on time, as calculating the features is very fast. In our tests no instance took more than one second for calculating the features. As for the GPHH approach, the training is will likely take a long time (hours to days of training depending on the instances used). The test, however, will likely only take a couple of minutes.

Chapter 4

Operator Selection and Bounds with Evolutionary Hyper-Heuristics

In this chapter we utilise a genetic algorithm-based hyper-heuristics to find effective and efficient local search configurations.

4.1 Introduction

Recent developments for the VRP have been driven towards complex and domain-knowledge-powered approaches [10, 176, 187]. Although these approaches have been more and more effective and efficient, current state-of-the-art metaheuristics need to apply several layers of domain knowledge/expertise to achieve good quality. This makes these algorithms hard to be replicable, time-consuming to design and very expensive due to this domain expertise. Moreover, it is unknown whether the existing manually designed heuristics can be generalised to different scenarios. When facing an unseen instance where the current best heuristic does not perform well, it is hard to manually design a specific effective heuristic again, as it highly relies on human expertise. The problem of selecting the best

heuristics for the given instance becomes a search problem in itself. When it comes to large-scale problems, these methods have a parameter-based and/or fixed limit on the search space, whether by experimentation or manually set. This limit can become a liability when searching for better solutions since the improving steps can be outside this restricted region.

Which bring us to one of the main techniques used to avoid those manually designed decisions: Hyper-Heuristics. HHs try to tackle the design level complexity by building a more general framework using low-level and easy-to-implement heuristics, usually having a trade-off on quality over time [28] [172]. However, the current HH studies do not employ pruning techniques at the lower-level heuristics (LLH) level, and therefore have limited scalability. When looking for LLH to apply in routing problems, even though HHs might efficiently look for the most suitable LLH, a conflict arises if simply applying these traditional pruning techniques to existing HHs and their LLHs: since these limits are manually designed, applying them would contradict the purpose of HHs. It can be argued that by doing so, we still reduce the amount of manual decisions — as the HH is still doing the high level design. However, as we want to improve the independence of the HH from expertise-based decisions we argue that it makes more logical sense to also have the search space be defined by the HH. Additionally, the manual limits are another optimisation task, since they can be either too small, avoiding finding better solutions, or too large and making it very slow. It would be fitting to give such a task to the heuristic space to control.

In other words, the goal of a HH is to find the optimal heuristic configuration, given a set of building blocks that solve a specific search problem. In other words, the HH will search for the best possible use of these blocks which will solve the problem. Therefore, a HH is theoretically able to find a configuration in the heuristic space better than (or similar to) a human-designed one, which is a possible definition of HHs as formalised by [150].

One of the most popular methods for HHs are based on Evolution-

ary Algorithm (EA) techniques. Probably the biggest advantage of EA techniques is their flexible representation scheme, which can be used to simplify the design of the search space. A Genetic Algorithm (GA) HH's chromosome provides a way to improve the order in which operators are applied through its evolutionary process, while also allowing for new elements to be explored.

Therefore, in this chapter we propose HH methods which consider automatically limiting the neighbourhood size generated by the LLH. We achieve this by automatically evolving the search space configuration with the help of a GA as our base HH algorithm. The GA, apart from its contribution to the search with its parallel concept, also permits a flexible representation.The GA chromosome can be modified to contain extra information, optimise the order and the selection of the operators, it can be evaluated differently and still be generic enough to be replicated for other problems (although we do not explore this attribute in our thesis due to being out of scope).

4.1.1 Chapter Goals

The key goal of this chapter is *to learn how to select the LLH operators and their limits to improve the search efficacy*. We achieve this by three different approaches. The first approach introduces an adaptive clustering technique which models the search space based on evolving clusters under a GA framework; the second approach introduces a new chromosome with an additional layer used to limit the corresponding operator search space; the third and final approach expands on the previous one by also considering an additional layer as well as some metaheuristic-specific parameters, including the initialisation heuristic. All of these methods contribute to reduce the search space while also selecting which operators to be used and in which order. This chapter has the following main objectives:

1. Show if the order of the operators matter and how they affect the

local search performance.

2. Provide an analysis on whether the traditional clustering techniques can be improved if they are adaptively changed through the runtime to better fit the problem being solved.

3. Propose a new GA chromosome that also limits the search space for each local search operator considered.

4. Present a GA chromosome that incorporates the different steps of the local search framework, including a new layer for further reductions of the search space, the initialisation heuristic and the penalisation.

4.1.2 Chapter Organisation

This chapter is organised as follows: Section 4.2 presents the cluster-based approach. In Section 4.3 introduces the dual-layered chromosome and the hyper-heuristic approach. Section 4.4 presents a third method which includes a rich chromosome that incorporates all stages of the KGLS. Finally, Section 4.5 summarises the chapter.

4.2 Cluster-based Hyper-Heuristics Approach

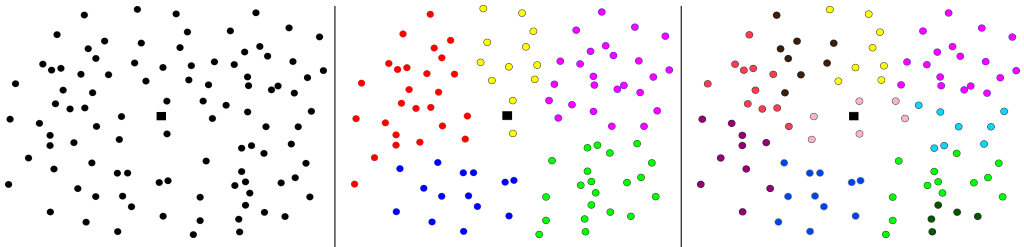
In this section we revisit one of the most famous techniques for reducing search space: *clustering*. The traditional approach utilises clustering techniques such as K-Means [126] to build a set of clusters of customers in which the routes can be contained in. This limits the search space size quite well, but also damages the ability of good solutions from being found, especially when these clusters do not allow exchange of information, making the quality of the solutions limited to the a random process (K-means is inherently stochastic). Not only that, once the clustering is done, it is usually fixed that way. This can be translated to sub-optimal searches, as

the optimal number of clusters might be lower or higher than the specified.

The K-means and other clustering techniques also try to have an even number of customers within each cluster, to which we have no proof that it is a good strategy and, perhaps more relevant, ignoring other features such as demand. We propose a new method which automatically updates the clusters based on the search directions. It can increase or decrease the number of clusters, changing them based on the routes being modified. We also allow information to be exchanged between clusters, thus avoiding a damaging reduction of the search space, while keeping it efficient.

The proposed Cluster-based Hyper-Heuristic (CbHH) is a HH based on a GA framework. It searches the heuristic space and also automatically trims the search space adaptively during the evolution of the solution. The CbHH does not require a sophisticated design and explores the solution space based on adaptive clusters. Additionally, two different types of chromosome decoding schemes are explored.

The idea is to initially create clusters based on the customers' location, and let the clusters evolve according to where the better solutions are heading to, as seen in Figure 4.1. In the example, there is a randomly created customer-distribution, Figure 4.1 (a), then the initial clusters are created (Figure 4.1 (b)), and as the evolution process progresses, the clusters will adaptively change, as in Figure 4.1 (c). These changes are guided through the use of local search neighbourhood moves, where we divide them into intra-cluster, where the moves only happen inside the cluster, and inter-cluster, where they happen between distinct clusters. A resulting route modification that affects both clusters will then trigger a cluster update. More details are given in the next sections. First, we present the solution chromosome.



(a) Original instance (b) Initial Clusters (c) Adapted Clusters as a result of the framework

Figure 4.1: A fictional example of how the cluster-based search adaptively changes the cluster space.

4.2.1 Solution Representation

In the CbHH, a solution represents a sequence of low-level heuristics that will be applied to improve an initial solution, this sequence represents the set of neighbourhoods to be explored in a local search framework. The GA chromosome used is a string of integers (as shown in Figure 4.2) where each allele represents the index of one of the LLH. Each value is unique for intra and inter clusters heuristics. In the Figure, for example, the value 3 can be a *Two-Opt** applied only inside each cluster, while the value 4 could be *Two-Opt** applied inter-cluster. Additionally, the chromosome has variable size, it can grow or shrink depending on the evolution process, which will be detailed in Section 4.2.3.

Figure 4.2: Example of a CbHH chromosome.

The LLHs are classical local search operators which are here divided into two groups based on how their neighbourhoods are explored: the first group considers the whole cluster of a given route, the intra-cluster group named as "*regular*"; and the second group will consider M closest clusters, here named "*cluster-based*", or the inter-cluster group. This division would

allow for the evolution process to determine when it is the best time to do exploitation within a cluster, while also allow for exploration between different clusters. Some of the LLH are utilized in both groups but are characterized by a different integer for the chromosome. Next, a list of these LLH, totalling 11 possible, as the ones in bold are utilized in both groups, otherwise, it is specified where they are applied.

- **Cross-Exchange**, swaps a sub-string of customers between two routes. Indexes 0 and 1, respectively for each type (intra and inter).
- 2-OPT, modifies two edges in a given route. Used only in the local cluster. Index 2.
- **2-OPT***, swap two edges in two given routes. Indexes 3 and 4, respectively for each type.
- OR-OPT, transfers a sub-string with a length of 2, 3 or 4 to another position. Used only in the local cluster. Index 5.
- OR-OPT*, transfers a sub-string with a length of 2, 3 or 4 to another route. Used only between clusters. Index 6.
- **MERGE**, combines two different routes into one, by concatenating one into any position of the other. Indexes 7 and 8, respectively for each type.
- **Reverse 2-OPT***, swap two edges in two given routes, but reversing the order of the remaining route. Indexes 9 and 10, respectively for each type.

Given this list of LLH and the chromosome structure, we can now explain the framework used.

4.2.2 CbHH Framework

The overall idea of the framework is based on the local search framework, but here we evolve the sequence of operators that will improve the initial solution, and also can change the search space on each iteration. Each operator in the chromosome sequence will consider the limits set by the clusters based on its group (*regular* or cluster-based), modifying the clusters according to the result of each move and repeating the process for each allele, always starting from the same initial solution given by the initialisation heuristic. The evolutionary process evolves the population of sequences. The final output of the framework will be the sequence of operators (and implicitly the sequence of cluster moves) which resulted in the best improvement. Figure 4.3 show the flow chart.

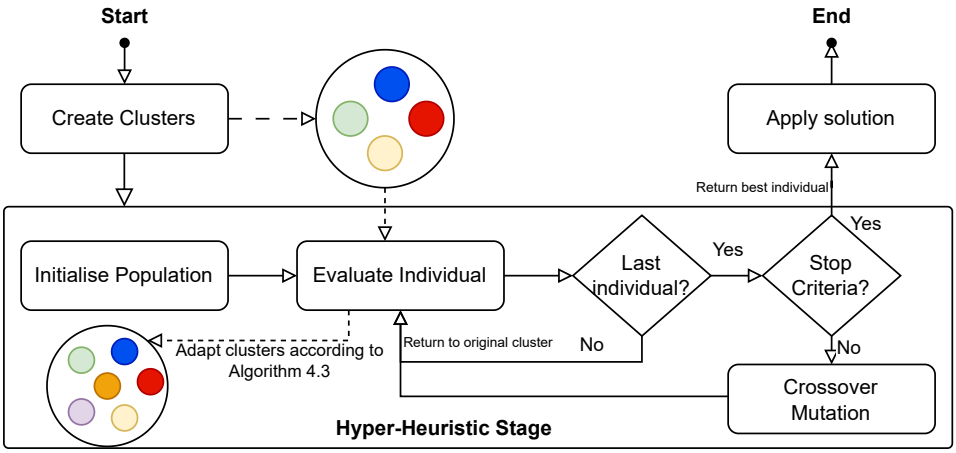


Figure 4.3: Overall flow chart of the proposed framework.

Following Algorithm 4.1, the initial clusters are created using the K-means algorithm (Line 2), a partitioning algorithm which is easy to implement, fast and efficient [98]. Then the initial routes are created respecting the clusters' limits (Line 3). For this initial solution, the savings algorithm of Clark and Wright (CW) [37] was selected, since it is a fast deterministic algorithm with fairly good results. Next, the evolutionary process (Lines 6-18) starts by creating the initial population (Line 4). The main loop starts

by selecting each individual in the population (Lines 6-7), and copies of the initial solution and clusters are created (Lines 8-9), only the copies are updated by the LLH. Then, for each allele in the individual (Line 10) the corresponding LLH is applied to the current solution (Line 11). The decoding process, shown in Lines 13-14, determines if the next LLH will be the first one or the next one, based on the type of decoding chosen. Here, decoding can be seen as the type of local search approach. We consider both the simple sequence of operators translated once or the Variable Neighbourhood Search (VNS), which changes neighbourhoods after each improvement (in this case goes back to the first allele). Then we verify if the current solution is better than the best-so-far solution, and replace it if the current solution is better (Lines 15-16). The crossover and mutation steps occur (Line 17). The loop is repeated until the stopping criterion is met (maximum number of generations), and then the algorithm returns the best individual (Line 18).

The clusters are updated according to the type of LLH applied, as shown in Algorithm 4.2. The cluster-based type is the one which explores more than the cluster for improvements and is composed of only inter-route LLHs. Their neighbourhoods consider the M closest clusters considering their centroids C_i (defined by Equations 4.1, 4.2 and 4.3, with Cl_i being the cluster i), where M is a percentage of the total number of clusters. For example, in an instance with 10 clusters, if the next selected LLH is Two-Opt* and $M = 20\%$, then for each cluster, the two (20% of 10) closest clusters will be considered. This is the main diversification step (other than the genetic process), since it allows for a larger, yet limited, neighbourhood to be explored. The clusters are updated based on the resulting route changes and whether they are majority inside a cluster or not.

$$\bar{x_i} = \frac{\sum_{j \in Cl_i} x_j}{|Cl_i|} \tag{4.1}$$

$$\bar{y_i} = \frac{\sum_{j \in Cl_i} y_j}{|Cl_i|} \tag{4.2}$$

Algorithm 4.1 Clustering-based Selection Perturbative Hyper-Heuristic

```
1: procedure CBSPHH(an instance, a list of LLHs)
2:     clusters ← Kmeans()
3:     initial_solution ← CW_Savings(clusters)
4:     population ← InitializePopulation()
5:     best ← ∅
6:     while Stopping Criteria not met do
7:         for each individual ∈ population do
8:             new_s ← initial_solution
9:             new_c ← clusters
10:            for each allele ∈ individual do
11:                new_s ← LLH(allele, new_s, new_c)
12:                Eval(new_s)
13:                if Improvement and VNS mode then
14:                    Restart from first Allele
15:                if Best Improvement then
16:                    best ← individual
17:        Evolve the population by Crossover and Mutation
18:    return best
```

$$C_i = (\bar{x}_i, \bar{y}_i) \tag{4.3}$$

Algorithm 4.2 Low-Level Heuristic Phase

```
1: procedure LLH(Integer allele, Routes solution, List clusters)
2:     repeat
3:         Limit search-space based on allele type
4:         Call allele function from LLH database
                    ▷ For example, if allele is Cross-Exchange cluster-based, invoke it.
5:         if allele is cluster-based then
6:             UpdateClusters(solution, clusters)
7:     until No improvement
```

The procedure, summarized in Algorithm 4.3, receives the clusters and the routes from a VRP solution. Then for each route, it verifies whether a new cluster will be created or just update the current ones. It does that with the function *ClusterWithMajority*, which returns the cluster in which

most of the customers belong to, if it has at least 50% of the route, otherwise returns 0. It can be noted that a new cluster will only contain the customer from one route, however in the next iteration of the selection process, new customers can be added (or even removed), dynamically changing the clusters and search space. As an example, suppose in a given solution, one of the routes starts at a given cluster **A**, passes through cluster **B**, and ends in cluster **C**, as a result of applying the Two-Opt* algorithm on cluster **A**, with **B** and **C** being considered neighbour clusters. And suppose this route contains 15, 14 and 17 customers in each cluster, respectively. Then this route will flourish as a new cluster **D** since the majority of the route is not within one cluster (the majority would be at least 50% of $15 + 14 + 17$). Another example, if some route had 10, 4 and 6 in each cluster, respectively, then the customers from **B** and **C** would become part of cluster **A**. If these customers were the only customers in either **B** or **C**, these clusters would be removed from the cluster list.

Algorithm 4.3 Cluster Update Phase

```
1: procedure UPDATECLUSTERS(Routes solution, List clusters)
2:     for each Route r ∈ solution do
3:         cluster ← ClusterWithMajority(r)
4:         if cluster = ∅ then
5:             RemoveFromClusters(r)
6:             AddNewCluster(clusters, r)
7:         else
8:             RemoveFromClusters(r)
9:             AddInExistingCluster(clusters, cluster, r)
```

4.2.3 Genetic Component

The Genetic Algorithm (GA) utilized is specified in this section. The algorithm follows the common GA framework where there is a string type chromosome, a crossover is applied, as well as a mutation step according to some rate. Each initial individual is created as a random permutation

that allows repetition of the LLH, with a fixed initial size but that can grow up to a δ **(delta)**, for example, if the initial size is 10 and δ is 5, the individuals can have anywhere from 10 to 15 alleles.

Two crossover operators are selected here. The first one is the same utilized in [85], called best-best, where for each parent the best-improving sequence of the chromosome is passed on to the children. The best-improving sequence can be determined by measuring how much each allele improves the solution, with the sub-string that provides the highest improvement being selected. For example, given an individual with 5 operators that has already been evaluated, if the sequence of improvements for the solution was $[3\%, 4\%, 5\%, 3\%, 2\%]$, then the largest improving sequence would be $[3\%, 4\%, 5\%]$. This sequence will be concatenated to the child part between the other parent's best-improving sequence, as denote by the black colour in Figure 4.4. The second one works similarly, but instead of the best-improving sequence, the cut points are at random, in a way to increase diversity. Each type has a 50% probability of being chosen. The top 10% individuals are saved for the next generation, while the remainder is composed by the resulting crossover of the current population. The parents' selection is done by random pairs, but one given parent can breed at most two times, being removed from the selection pool after the second time, to avoid oversampling of a given scheme.

For the mutation step, similarly to the crossover, two methods are applied with a $65 : 35\%$ chance, if the mutation occurs. The first one removes the worst improving sequence of a chromosome, called remove-worst, also from [85]. The remove-worst works in the opposite way of the best-improvement, where the sub-string with the least contribution to the solution's quality gets removed. For example, considering the sequence of improvements to be $[3\%, 0\%, 5\%, 0\%, 0\%, 6\%]$, by removing the worst improving sequence we would get $[3\%, 0\%, 5\%, 6\%]$. The second mutation operator randomly adds new LLH, up to one fifth of the selected chromosome length, at random positions, increasing diversity. After the

crossover step, a mutation step with a 10% rate is applied. Additionally, and to avoid falling in a local optimum schema, half the population is reset (a new random permutation) every 20 continuous generations without improvement, where the 50% best individuals are kept (considering their fitness value).

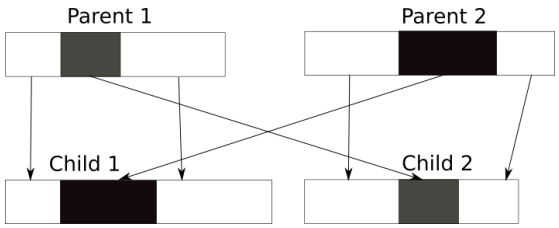


Figure 4.4: Example of a best-best crossover, based on [85].

Since each individual in the population will have a different order in which the LLH are applied, there is a second implicit diversification step (other than the cluster-based LLH mentioned above). To explain a bit further, with a different order in which the LLH are applied, then the changes in clusters and in routes will ripple into the next selected LLH. Because applying a cluster-based Cross-Exchange first into the initial solution and then applying a Two-Opt in each cluster will most likely result in a different set of routes and clusters than if doing the other way around (Two-Opt first and CE second). Therefore, the method explores the heuristic space while also exploring the solution space.

4.2.4 Experiment Design

In this section, the experiment details are described. The algorithms compared in this experiment were coded in C++[1] with the library VRPH (created by [82]), from which the VRP basics and some of the implemented heuristics were directly reused. The experiments were run on computers with an Intel®Core™i7-8700 @ 3.2GHz and 15GB memory.

[1]Version 11 compiled with g++ and optimisation flag -O2.

4.2.5 Experiment Goals

The objective of these experiments is to validate the idea of the Cluster-based Hyper-heuristics. For this purpose, four types of experiments were conducted, all considering the LSVRP:

- **Experiment 1**: the goal is to show the effects of changing the search space with the parameter M (the percentage of neighbour clusters which are considered in a search), by considering a larger or smaller size, and to compare with fixed clusters, or no clusters at all. This experiment was conducted by having three different instances being run varying the parameter M to 25% and 50% (0% would not consider inter-routes between clusters and therefore was not tested, and 100% is the same as no clusters since all are considered). Expectations are that with a larger number of neighbouring clusters, the solution will improve at the cost of higher computation time, and the exact opposite for a lower value. However, by how much, and what point has the best trade-off between quality and execution time?

- **Experiment 2**: this experiment investigates how much the starting clusters affect the solution. To achieve that, two approaches are chosen: the first one is a route-first cluster-second approach, where the K-means is used after the initial routes are created, this way the starting clusters will have multiple routes passing and the cluster evolution will happen accordingly; the second is a cluster-first route-second, where the K-means is run 30 times (since the clusters depend on the initial random points, they can have different outcomes) and the one having the best initial solution is chosen (creating the clusters first, will make the initial routes obey the cluster delimitation).

- **Experiment 3**: it shows the effects of different chromosome decoding approaches. The objective is to show by how much solution quality changes from a VNS-based type decoding to a single pass, and how is the execution time affected.

- **Experiment 4**: it compares the Cluster-based HH solutions to the best-known solution, as well as to a deterministic algorithm (CW) and to a manually designed combination of the same LLH that we built for this experiment. The objective of this experiment is to show whether the proposed method is competitive to existing methods, in both solution quality and execution time. This experiment utilizes the best configuration found by experiments 1 and 2, aiming to be the most competitive.

The results and discussions regarding these experiments are given in Section 4.2.7. But first, the dataset and parameters are shown.

4.2.6 Dataset and Parameters

The instances utilized are a sub-set from the CVRPLIB dataset of [183], which ranges from 100 to 1000 customers, varying both geographical distribution and vehicle capacity. The size of these instances meet the large-scale definition and have known optima or a good solution is known.

Unless otherwise mentioned in a given experiment, the parameters use the default values described in Table 4.1.

Table 4.1: Default parameters.

Parameter Name	**Value**
M (for closest clusters)	25% of the number of clusters
K (for K-means)	10% of instance size
VNS Type	Explore Sequence
Generations	100
Population size	30
Minimum individual size + δ	$22 + 11$
Number of runs	10

4.2.7 Results and Discussions

This section discusses the results of the experiments described above.

Experiment 1

This experiment considers the effect of changing the search space, as well as comparing it to its counterparts: not using the cluster limit on the search, and *not* updating the clusters in the proposed method. Table 4.2 presents the results, showing for each case the average solution with standard deviation, the best result and the average execution time in minutes. As expected, the cases in which the search space includes more clusters (such as No Clusters and 50%) take longer, since the number of evaluations needed is significantly higher, while the quality increase is not much different. The fixed clusters also have a worse performance overall, since the search space is likely larger than the proposed approach, and it avoids different search areas, searching the same clusters throughout execution.

Table 4.2: Results for Experiment 1: the effectiveness of the proposed method when changing the search space size. The average solution, as well as the standard deviation and best solution, are shown, followed by execution time in minutes.

	No Cluster			Fixed Cluster (25%)		
Instance	Avg(s.d.)	Best	Time	Avg(s.d.)	Best	Time
X-n200-k36	60898(429)	**60537**	216.6	61024(460.22)	**60537**	126.84
X-n251-k28	40401.5(96.62)	40266	250.68	40298.15(173.65)	**39847**	47.51
X-n308-k13	27580(155.32)	27361	1032.72	27736.65(187.08)	27343	170.40
	CbHH (25%)			CbHH (50%)		
Instance	Avg(s.d.)	Best	Time	Avg(s.d.)	Best	Time
X-n200-k36	60996.89(194.96)	60600	**6.53**	60826.50(368.22)	60543	40.84
X-n251-k28	40339(129.52)	39974	**8.48**	40428.41(146.38)	40195	73.83
X-n308-k13	27973.73(386.11)	27205	**33.66**	27353.5(160.12)	**27016**	297.8 3

Experiment 2

In this experiment, the starting solution is explored. As shown in Table 4.3, the route-first solution is worse than its counterpart. One possible explanation is that the initial Savings algorithm will make the clusters adapt to its routes and reach a local optimum much faster. In fact, all converge to the same solution. While for the cluster-first the initial routes are freer to be updated.

Table 4.3: Results for Experiment 2: a comparison between both methods of initialization. The solution average and standard deviation are shown.

	Route-First	Cluster-First
Instance	Avg(s.d)	Avg(s.d)
X-n200-k36	60993(0)	60996.89(194.96)
X-n251-k28	40406(0)	40339(129.52)
X-n308-k13	28242(0)	27973.73(386.11)

Experiment 3

This experiment investigates the difference in time and quality based on how the chromosome is decoded. Table 4.4 shows the effects of different decoding of the chromosome on quality and performance for the given instances. The VNS is expected to have a better result since it explores the search space more heavily, at the cost of more time. However, the difference between the two is smaller than expected, with a GAP of less than 2% between the two modes. Less surprisingly, it finds the best solution in all test cases. We can also notice that the extra computational cost is higher on the smaller instances when compared to the sequential decoding.

Table 4.4: Results for Experiment 3: a comparison between the two types of decoding for the chromosome. Showing the average, standard deviation and execution time in minutes.

	Explore Sequence		VNS	
Instance	Avg(s.d)	Time	Avg(s.d)	Time
X-n200-k36	60996.89(194.96)	6.53	61034(358.25)	33.91
X-n251-k28	40339(129.52)	8.48	40277.88(166.18)	15.48
X-n308-k13	27973.73(386.11)	33.66	27435.82(117.88)	35.49

Experiment 4

This final experiment compares the proposed method with the best-known solution [2], a manually designed combination of the LLH and to the Savings algorithm. The manually designed version is based on creating chromosomes of fixed length which uses all the selected LLH (therefore of length 11), chosen in random order, and are not put through the GA process (all other parameters are the same as in 4.1). This aims to simulate the process of selecting the LLH and using them in some specific order, which is commonly done in heuristics design. Table 4.5 summarizes the comparison. The proposed method performs better than the other two and is within 5% of the best-known solution for all instances.

Overall the results seem promising in showing that having the adaptive clusters improve the search speed, without much loss of quality. It performs better than some of the compared cases. We performed a Wilcoxon Ran-Sum test to verify the statistical significance of this performance comparing the base case with the tested variants: the fixed clusters, regarding parameter M, cluster-first and decoding type VNS. The results, as seen in Table 4.6, show that the method is significantly better for most cases.

[2]Taken from the CVRPLIB website (`http://vrp.galgos.inf.puc-rio.br/index.php/en/`)

Table 4.5: Results for Experiment 4: comparison between different BKS and different algorithms with the proposed method. The best, the average with standard deviation, as well as the GAP from the best solution to the BKS are shown. BKS with (*) indicate that they are the optimal.

		Manually Designed			Savings-CW		Cluster-based HH (25%)		
Instance	BKS	Best	Avg. (s.d)	GAP	Cost	GAP	Best	Avg. (s.d)	GAP
X-n200-k36	58578*	61220	61892.17 (437.72)	4.5%	61167	4.4%	60600	60996.89 (194.96)	3.4%
X-n251-k28	38684*	40846	41096.2 (250.2)	5.5%	40576	4.9%	39974	40339 (129.52)	3.3%
X-n308-k13	25859	27970	28106.4 (136)	8%	28555	10.4%	27205	27973.73 (386.11)	5%

Table 4.6: Wicolxon Rank-Sum test: p-values of the comparison between the base method (CbHH 25%) and the other evaluated variations.

Instance	Fixed Cluster (25%)	CbHH50	Route-First	VNS
X-n200-k36	0.27	0.01	0.82	0.55
X-n251-k28	0.24	0.04	0.06	0.06
X-n308-k13	0.006	0.00	0.01	0.00

4.2.8 Summary

In this section we explored our Cluster-based Hyper Heuristics. The method focus on simple local search methods to measure the efficiency of different neighbourhood orders and our novel adaptive clustering approach. The clustering technique applied evolves their distribution with each iteration — including the number of clusters. We show that this method is able to find more efficient search when compared to the traditional fixed approach. Additionally, even though the evolution process can still be considered slow when compared to other meta-heuristics, this can be done offline, and the application of the resulting chromosome takes very little time (less than a minute for all instances).

4.3 The Dual-Layered Chromosome GA

This section presents our second Selection HH. This approach considers automatically limiting the neighbourhood size generated by the LLH, based on an evolutionary process and a new chromosome. The idea is to let each LLH have its own neighbourhood size (i.e. subset of neighbouring customers to be examined) adaptively evolved according to its effectiveness in the evolution, which leads to an adaptive search space for each LLH. The goal is to investigate whether HHs could automatically find an adaptive neighbourhood size strategy that can successfully find quality results, especially when compared to fixed ones.

This method utilizes an Evolutionary Hyper-Heuristic framework based on a Genetic Algorithm to evolve the selection of LLH available to the best order and their limits, up to a fixed number of generations. But before going into more details, we present the solution representation.

4.3.1 Solution Representation

The solution chromosome utilizes a two-dimensional array with variable length, where the first level is an integer ID which represents a given LLH, and the second level is the percentage of the closest neighbours that the LLH will search for. Even though all alleles have the two levels, the search space limit is only applied for inter-route neighbourhoods. This is because an intra-route operator will only perform the search within the same route, which is usually a very short number of customers due to the capacity constraints. In inter-route operators, however, all other customers can potentially be searched for a move. In a large scale problem the number of customers outside the route can be quite large, making it a very expensive evaluation, which is why we want to limit only these operators. The limits are still kept for all alleles because crossover operations can alter the structure of the alleles in such a way that an intra-route allele might affect an inter-route one. These operators are further explained later.

For the utilized LLH, we have selected a set of intra and inter perturbative operators. Some are very frequently used and historically show good improvements, such as Two-Opt and Cross Exchange. But more importantly, they are all relatively easy and straightforward to implement and use, which is one of the arguments of using Hyper-Heuristics [28]. The LLH used are the same set as the previous method, presented in 4.2.1, but without the clustering-related differences, i.e. just the original implementations. Figure 4.5 is an example of the representation. As the ID 3 in the first position of the first dimension represents a two-opt* move, then it will consider the closest 50% of all customers to perform the move; the next ID 3 represents the same operator, but now only considers 30% of them.

3	1	5	2	0	4	1	3	0
50%	30%	20%	20%	50%	20%	10%	40%	30%

Figure 4.5: An example of chromosome for the Genetic Algorithm.

4.3.2 Evolutionary Hyper-Heuristic

The hyper-heuristic framework has an evolution process where each individual, represented by the above mentioned two-dimensional array, will apply its sequence of LLH to a single starting solution. Algorithm 4.4 summarizes the process. First an initial solution is created heuristically (line 2), using either the Savings heuristics [37] or a simple round-way trip for each costumer, both cases are tested to see which performs better. Next, the initial population of heuristics, each represented as a LLH sequence, is created at random (line 3). In an individual, each LLH is associated with the percentage of closest neighbours it considers. The main loop starts (line 5) and each individual will be evaluated by how much they improve the initial solution. Each allele and its correspondent LLH will modify the current solution if that improves it (lines 8-9), i.e, the acceptance criteria is

Only Improvement, as shown in Algorithm 4.5. If the solution of the current individual is the best found, we save or update that individual (lines 10-12). Finally, if the stopping criteria are not met (line 13), the crossover and mutation operators are called to evolve the population (line 14). The algorithm returns the result given when solving the instance with the best individual (line 16).

Algorithm 4.4 Adaptive-based Selection Perturbative Hyper-Heuristic

```
1: procedure ABSPHH(Dataset instance, List LLH)
2:     initial_solution ← FindInitialSolution()
3:     population ← InitializePopulation()
4:     best ← ∅
5:     while Stopping Criteria not met do
6:         for each individual ∈ population do
7:             new_s ← initial_solution
8:             for each allele ∈ individual do
9:                 new_s ← LLH(allele, new_s)
10:            Eval(new_s)
11:            if Improvement then
12:                best ← individual
13:        new_population ←top 10% individuals
14:        while new_population ≠ Full do
15:            Select 2 parents at random from population
16:            new_individuals ← Crossover selected parents
17:            Mutation of new_individuals according to probability P_M
18:            new_population ← new_individuals
19:        population ← new_population
20:    return best(instance)
```

The evolutionary operators are responsible for improving the individuals over the generations. They will diversify the population and try to exploit the best individual characteristics. Before this process, however, elitism is applied to the top 10% best individuals, which are kept for the next generation. The remainder of the population is replaced with the children. The parents selection occurs by random chance, but no individual

Algorithm 4.5 Low-Level Heuristic Phase

```
1: procedure LLH(Array allele, Routes solution)
2:     repeat
3:         Limit search-space based on allele[1]
4:         Call allele[0] function from LLH database, update solution accordingly  ▷ For example, if allele[0] is Cross-Exchange invoke it.
5:         solution ← correspondentLLH()
6:     until No improvement
7:     return solution
```

can be chosen more than twice.

For crossover, similarly to our previous method, two operators were chosen aiming to create different diversification aspects, but this time they have the same probability of happening. The first operator is the classical two-point crossover, where two points are chosen at random, and the middle portion of each parent is swapped between both to generate the offspring. For this operator, the solution search space limit at the second dimension of the resulting array will be updated by the average between the parents' limits, considering the same positions, as shown in Figure 4.6. If a child has a larger size, the original values are kept. This operator is mainly used to produce some diversity in the population by considering new neighbourhood limits and by not considering the solution quality. The second operator is the Best-sequence crossover, based on [85]. The

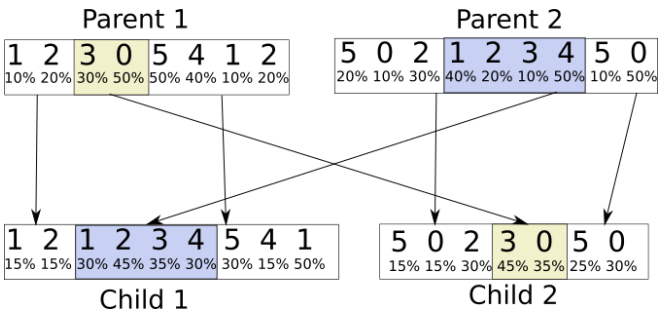


Figure 4.6: Example of a two-point crossover for the two-dimensional chromosome.

best improving sequence from both parents, i.e. the largest subset of consecutive operators that contribute the most for the solution, are inherited by the children. It works similar to the first operator, but the average is not applied. Instead, the sequences are copied exactly the same, since its goal is to keep the good parts of the solution.

Mutation has a probablity $P_M = 10\%$ chance of happening for each individual of the new population, and also has two operators with equal chance of being chosen. The first one, also presented in [85], removes the worst performing consecutive sequence of operators from the individual which do not contribute to the solution, as used in our previous method (seen in Section 4.2.3). The second one adds new random operators from the LLH pool as if newly created, which can vary between 1 and up to 20% of the length of the chromosome, and can be added in any place at random. These mutation operators aim to remove the bad performing schemes from the population and to add new elements for diversification. The two operators are illustrated in Figure 4.7.

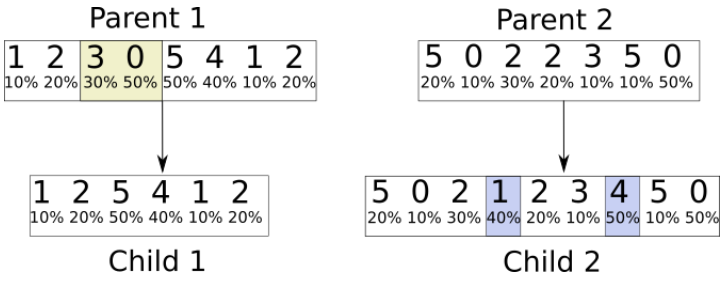


Figure 4.7: Example of the mutation operators. On the left, the worst improving sequence is removed. On the right, new random LLHs are added.

Note that the order of the operators will result in different final solutions. But also the limits play a role in this, since even having the same operators sequence, the search limits may be different and will result in a distinct solution.

4.3.3 Experiment Design

The experiments were designed aiming to test the method's ability to limit the search space. However the evolution can be sensitive to the initial solution and a test was designed to measure this. We also want to show whether there is a trade-off between the size of the limits and the quality of the final solution. We designed three main experiments for this preliminary work, which are detailed next.

Experiment 1 measures the impact of different initial solutions on the performance of the HH approach. As traditional perturbation operators might not escape from local minimums too easily, the initial solution can significantly change the outcome. We test an obvious feasible approach which is a round trip between the depot and all customers, and compare it with the traditional Savings from Clarke and Wright [37]. The Savings algorithm is a deterministic constructive heuristic which starts building routes by connecting different round trips, whose connection will bring the most savings. This test will also identify wheter the chosen simplistic LLHs are enough to produce sufficient improvement without any local minima escape mechanism given the two different starting points.

Experiment 2 aims to evaluate both the efficiency and performance of the adaptive limits to the search space when compared to a fixed limit. To achieve that, we apply the same Hyper-Heuristics framework but set a fixed limit to the neighbourhood search. The fixed limit used as rule of thumb has all alleles set to 25%. On the other hand, the adaptive limits are randomly chosen for each LLH, ranging from 10% to 50%. For this experiment we assume that the full neighbourhood (i.e, 100% size), would not find results a lot better given the extra amount of time necessary to do so. Hence this experiment aims to show how having a fixed neighbourhood size (like most works do) compares to having different limits throughout execution. Since we have only a set of few LLHs and are comparing this fixed size based on our own framework, all claims made here would be true at least considering the tested circumstances. But at least if finding

positive results we can show that this idea is worth investigating. For example, with this experiment we can analyse whether our HH would always give priority to the LLHs which have a larger neighbourhood, which could be expected since they will cover more possibilities.

Finally, **Experiment 3** compares the proposed method with two manually designed heuristics, as well as the best known solutions. Two manually designed heuristics, which use the limits to the search space as proposed in this paper, were created based on intuition from the authors. Both share the same work process, listing all operators used in our LLH pool twice. For example, with the six operators used numbered from 1 to 6, the manually designed heuristic would be [1,2,3,4,5,6,1,2,3,4,5,6]. Then, we apply it to the initial solution generated by the Savings heuristic [37] in this fixed order. The difference between the two manually designed heuristics are obtained by changing the neighbourhood limits between the two groups of operators. The first heuristic simulates a top-down approach, where we apply broader (50%) search spaces to the first group of operators and narrower (10%) to the second. The second manually designed heuristic is the bottom-up approach, which is the exact opposite of the previous one, applying the narrower size for the first group of operators, and broader size for the second. We will use **MHTD** and **MHBU** when referring to each heuristic, respectively, in the next sections.

The program was coded in C++[3] with the VRPH library [82], from which the VRP structure and some of the implemented operators were used. The experiments were run on a Intel®Core™i7-8700 @ 3.2GHz and 15GB memory. Table 4.7 shows the parameters of the Genetic Algorithm. Since the individual size has a variable length, the δ value indicates the upper bound for the initial population chromosomes. Additionally, in order to speed-up the process and escape local minima, if the GA finds no improvement for 10 consecutive generations, the lowest performing half of the population is replaced by randomly generated individuals, and if the

[3]Version 11 compiled with g++ and optimisation flag -O2.

lack of improvement persists for another 10 generations, the main loop is finished. Next we present and discuss the results.

Table 4.7: Genetic Algorithm parameters.

Parameter Name	Value
Generations	100
Population size	30
Initial chromosome size + δ	$18 + 9$
Crossover rate	Every generation
Mutation rate	10%
Elitism	10% top rated
Number of runs	30

4.3.4 Results and Discussions

All the experiments were conducted by using a sub-set of instances from the CVRPLIB proposed by Uchoa et al [183], which vary between distribution, size, demand and capacity.

4.3.5 Effect of initial solution

The first experiment aims to show the impacts of initial solution to the algorithm convergence. As shown in Table 4.8, both the Round-Trip and the Savings heuristic [37] were used as initial solution. To verify if the results are significant, a Wilcoxon rank-sum test was also performed comparing the two approaches. If the *p-value* is less or equal to 0.05 we say that the algorithm is statistically better when solving that instance. The instances in which there are significant advantage are marked in the table with (+). For this case, all results are significantly better, when considering Savings as the initialisation heuristic.

Analysing Table 4.8, the impact of the two different starting solutions is more clearly noticeable when looking at the execution time. This is be-

Table 4.8: Results comparing the initial solution impact. Execution time in minutes

	Round Trip			CW Savings [37]		
Instance	Initial Solution	Avg(s.d)	Time	Initial Solution	Avg(s.d)	Time
n200-k36	295558	61205.2(124.66)	31.39	61167	60993(0)(+)	15.11
n251-k28	290890	41045.73(200.84)	55.82	40576	40385(0)(+)	22.29
n308-k13	410930	27453.33(129.41)	191.77	28555	28040.3(7.97)(-)	49.80
n351-k40	28240	28049.7(184.59)	200.54	27123	26978(0)(+)	52.20
n376-k94	558384	151778.77(430.00)	487.78	149659	148957.2(8.26)(+)	279.02
n401-k29	755820	69031.8(311.91)	250.22	68975	68666.1(84.26)(+)	90.81
n420-k130	318530	113619.77(332.58)	1002.74	112604	112286.63(29.26)(+)	517.06
n449-k29	704352	59424.9(285.00)	322.63	58614	58395(0)(+)	75.60

cause Savings can provide a much better initial solution than Round Trip, making the fitness evaluation (improving the solution until convergence) much less time consuming. In addition, as can be noticed by the average and standard deviation, the Savings solution tends to converge to a single point or a very close one. This means most solutions reach the same local minima when considering the used LLH.

On the other hand, the round trip version does not converge to a single point and has a greater relative improvement. This behaviour allows us to investigate the use of the neighbourhoods limits more clearly, as it provides a longer range of possible solutions. Therefore, next tests shown will utilise this method for generating the initial solution.

4.3.6 Effectiveness of the adaptive size limit

The second experiment is focused on showing how the proposed chromosome and evolutionary strategy performs for different search space limits, given the same Hyper-Heuristic framework. Considering a fixed neighbourhood search limit of 25% and the adaptive one, which range from 10% to 50%. This experiment aims to test if they have significant difference regarding both quality and execution time. Since starting with the Savings heuristic led to prematurely convergence (as seen in the previous

experiment), this test will use Round Trip starting solutions.

Table 4.9 summarizes these results. Although the adaptive limit does not outperform the fixed one in every scenario, the larger instances show better results. We also compare the p-value given the Wilcoxon rank-sum statistical test. Like in the previous experiment, the (+) sign means that the results are significantly better. The (=) sign means that there is no clear advantage when using the approach and the (-) sign means it performs worse. As the table shows, the adaptive method outperforms the fixed one for most instances, especially in larger scenarios. Although there is a somewhat significant increase in time, we deem this result as encouraging to keep exploring adaptive-based limits. Additionally, the ones that the adaptive method do not outperform, can also be an indication that our evolutionary mechanisms could not find solutions with similar (or better) schemes of limits.

Table 4.9: Results comparing a fixed search space size and an adaptive one. Execution time given is in minutes

	Fixed size (25%)			Adaptive size		
Instance	Best	Avg(s.d)	Time	Best	Avg(s.d)	Time
n200-k36	**60659**	61270(304.61)	39.19	60959	61205.2(124.66)(=)	31.39
n251-k28	40560	41022.76(204.00)	62.34	**40512**	41045.73(200.84)(=)	55.82
n308-k13	**27041**	27331.26(102.96)	210.52	27150	27506.76(151.49)(-)	191.77
n351-k40	27748	28240.63(360.46)	187.08	**27641**	28049.7(184.59)(=)	200.54
n376-k94	**150591**	152397.93(1009.14)	502.00	150715	151778.76(430.00)(+)	487.78
n401-k29	68484	69303.18(403.75)	274.30	**68466**	69031.8(311.91)(+)	250.22
n420-k130	112976	114116.67(420.37)	731.18	**112963**	113619.77(332.58)(+)	1002.74
n449-k29	59075	59773.9(351.28)	270.56	**58783**	59424.9(285.00)(+)	322.63

4.3.7 Comparison with other methods

The third experiment compares the proposed approach with the Best Known Solution[4] (BKS) and with the two manually designed heuristics specified in previous section, MHTD and MHBU. The solutions generated by the Savings algorithm [37] are shown as a comparison baseline. The proposed approach, as does the manually designed heuristics, uses the Round-trip as initial solution, where each customer is served by a different vehicle. The adaptive search approach is described in Section 4.3.2, and range of neighbourhood limits is between 10%-50%.

Table 4.10 shows these results. When compared to BKS the proposed solutions are significantly close, specially considering our approach does not have local optima escape mechanisms. This indicates that the neighbourhood limits can be effective regarding solution quality. The better or similar performance by the Savings algorithm, however, proves that the starting point is far too bad to lead to better solutions given the chosen parameters and LLH. The results also show that the proposed method does have an overall better solution quality than the manually designed heuristics which have fixed search limits, outperforming them for all cases, even in the average case. Therefore, this suggests that having the different order and search sizes improves the search space exploration.

4.3.8 Further Analysis of the evolved solutions

In this work, we want to show whether the automatic evolution of the search space size will give benefit to the overall search. Experiments 2 and 3 measure this more directly, while experiment 1 tries to show whether the initial solution is the only factor making the results seem like they are. As shown in Experiment 2, some of the fixed size search performed just as good as the proposed approach. However, this can be explained if we

[4]Collected from the CVRPLIB webiste (`http://vrp.galgos.inf.puc-rio.br/index.php/en/`)

Table 4.10: A comparison between BKS and the manually designed heuristics with the proposed method. The GAP shows how close to the BKS the average solution is. BKS with (*) indicates that they are the optimal. The best solution for each instance is highlighted. The Savings algorithm from [37] (CW) is shown as a point of comparison.

		MHTD		MHBU		CW [37]		Proposed Method		
Instance	BKS	Best	GAP	Best	GAP	Best	GAP	Best	Average(sd)	GAP
n200-k36	58578*	62306	6.3%	62067	5.9%	61167	4.4%	**60959**	61205.2(124.66)	4.4%
n251-k28	38684*	41787	8.0%	41960	8.4%	40576	4.8%	**40512**	41045.73(200.84)	6.1%
n308-k13	25859	29012	12.2%	28374	9.7%	28555	10.4%	**27150**	27506.76(151.49)	6.3%
n351-k40	25928	29253	12.8%	28263	9.0%	**27123**	4.4%	27641	28049.7(184.59)	8.1%
n376-k94	147713*	159458	7.9%	152638	3.3%	**149659**	1.3%	150715	151778.76(430.00)	2.7%
n401-k29	66187	73098	10.4%	70183	6.0%	68975	4.2%	**68466**	69031.8(311.91)	4.2%
n420-k130	107798*	115910	7.5%	115640	7.3%	**112604**	4.4%	112963	113619.77(332.58)	5.4%
n449-k29	55269	61982	12.1%	60376	9.2%	**58614**	6.0%	58783	59424.9(285.00)	7.5%

analyse the evolved heuristics. As shown in Table 4.11, the evolved best individual for each case has an average of their search space limits around the 29%, meaning the fixed approach was actually a good approximation of this. Figure 4.8 shows one run's best individual for instance *n308-k13*.

This also gives insight on how larger neighbourhoods might not always translate to better solutions. Other work might use smaller neighbourhoods assuming they get a time benefit in exchange for some reduction of the overall quality, but these results actually tell otherwise. As these evolved limits barely do not converge to using the largest available limit, since we are only measuring quality. In fact, as already mentioned, on average all best individuals limits' are around 29%, when considering all non-fixed solutions. Additionally, on average only 29% of the chromosome have a limit of 40% or higher (up to 50%), considering the same best individuals. This means that the evolution process does not prefer larger neighbourhoods, and uses them only for around $1/3$ of the alleles. And around 35% of the chromosomes have neighbourhood sizes of equal to or less than 25%. This means that the evolution process brings balance between large and small neighbourhoods.

Table 4.11: The average neighbourhood limit for each Low-Level Heuristic and the average in total for each given instance.

Instance	Two-Opt	Or-Opt	Cross-Exchange	Two-Opt*	Or-Opt*	Merge	Total
n200-k36	30.08%	30.32%	29.96%	29.13%	30.46%	27.39%	29.52%
n251-k28	30.36%	29.60%	30.18%	30.50%	29.67%	28.93%	29.99%
n308-k13	30.54%	30.45%	29.32%	29.79%	31.07%	28.57%	29.93%
n351-k40	29.23%	30.10%	30.25%	29.51%	30.74%	26.03%	29.39%
n376-k94	28.92%	30.57%	29.89%	30.26%	30.78%	26.44%	29.46%
n401-k29	30.64%	30.34%	30.19%	29.49%	29.51%	26.85%	29.70%
n420-k130	29.78%	29.75%	30.59%	29.78%	32.41%	26.63%	29.62%
n449-k29	28.66%	29.91%	30.25%	30.51%	30.88%	27.40%	29.69%

2	3	3	2	5	0	2	3	2	0	5	0
38.7%	34.6%	10%	43.1%	16.8%	27.6%	20.6%	34.2%	27%	35.3%	50%	30%
4	2	2	4	5	0	1	0	5	0	5	1
26.9%	23.5%	22.6%	40.6%	27.1%	40%	30%	26.2%	41.2%	33.7%	23.1%	26.5%
2	2	4	2	0	0	1	1	0	1		
30%	20%	25%	10%	25.6%	28.7%	35.3%	31.2%	32.1%	20%		

Figure 4.8: Example of an evolved chromosome.

4.3.9 Summary

In this section we present our novel dual-chromosome-based hyper heuristic. The dual-layered chromosome has two objectives: find the optimal order of the neighbourhood operators used and their respective search sizes. We also introduce a crossover and mutation operator specific for this new chromosome. The method is able to find better solutions than a manually designed strategy. Another finding regards the pattern among the evolved heuristics, which do not give preference to larger limits over smaller ones. This can indicate that the search limits actually are an effective way of improving performance of LSVRP algorithms, since they improve efficiency (when compared to full neighbourhood size) without loss of quality. However, there are still some questions raised from look-

ing at results. For example, since one of the possible permutations for the adaptive approach can be the same as the fixed one, then how did it not find something similar? Perhaps the genetic operators did not allow the adaptive search to find the same (or better) limits.

4.4 Evolutionary Hyper-Heuristic for Knowledge-Guided Local Search

Our third proposed approach applies an EHH to optimise the order in which simple operators are used to solve the LSVRP, reducing the pre-configuration required which is usually done by an expert. The evolutionary process tries to find the best neighbourhoods to be used and their sequence, as well as the solution construction method and local optima escape mechanism. Additionally, the EHH also evolves the heuristic pruning process through a new chromosome, letting the evolutionary process determine the best heuristic configuration.

In this section, we describe the Evolutionary Hyper-Heuristic (EHH) developed. The overall idea is to use a Genetic Algorithm (GA) to evolve three different parts of the KGLS, which is based on local search and has its phases clearly distinct, and also the order in which the operators are applied (as a Selection Hyper-Heuristics), together with how much of the neighbourhood should be searched for each operator, and the order of penalization criteria. Although these other parts are related to other steps in the local search framework, we still consider its main contribution in the improvement step.

4.4.1 Representation

The chromosome for this GA has three parts, as shown in Figure 4.9. The first part of the array, or **Initialisation part**, decides which initialisation heuristic to use. The integer value represents the algorithms that will

build the initial solution. We consider a Random initialisation, the Savings heuristics from [37], and a Round-Trip (all customers are served by one vehicle each), with values $1, 2, 3$, respectively.

Figure 4.9: Chromosome for our evolutionary hyper-heuristic.

The second part, or **Optimisation part**, utilises a three-layer array: the Neighbourhoods layer, the Limits layer and the Decay layer. The first layer represents the sequence in which the operators are to be used. This layer searches for the most effective order in which to apply the operators to all instances through the evolutionary process. This concept has been applied to other problems before and have shown competitive results, such as [43, 85, 100, 159] and was explored for the VRP in [42]. The second layer, or Limits layer, contains the limits applied to the number of customers to be searched, ranked by the closest ones for every single customer. For example, if the operator has 30% in the second layer, it means that only the 30% closest customers can be considered for the move. The Decay layer also trims the search space by reducing the number of routes that can be evaluated in inter-route moves. For example, if the third layer has a value of 50%, the correspondent operator will only consider 50% of all routes which are closest to the current one being evaluated. This closeness is calculated by the distance between the route's centre of gravity.

For the Optimisation part, the operators are easy-to-implement traditional VRP ones. They are, with index (bold ones are inter-route):

1. Swap: Swaps position of two customers within the same route.
2. Two Opt: Exchanges two edges within the same route.

3. **Cross Exchange**: Exchanges two sub-routes of size k between two routes.

4. **Swap***: Swaps two customers on different routes.

5. Three Opt: Exchanges three edges within the same routes.

6. **Relocate**: Moves a customer from one route to another

The **Penalisation part** is the final part of the chromosome and represents the order of the three badness functions in which the KGLS penalises the solution (which are explained next). These are mapped as integers, each representing one of the three functions (Equations 4.4, 4.5 and 4.6). However, these badness functions are not limited to inter-route moves only, nor does the heuristic follow the same pattern of inter to intra, as the original work does. This flexibility allows for more degrees of freedom regarding how to apply the penalisation.

4.4.2 The KGLS badness functions

We already explained the KGLS [9] at Chapter 2, but we recall some of its functions here for an easier connection to our chromosome.

The functions $b(.)$ measure the badness of an edge in the penalisation phase of the KGLS. The original work utilises three different functions $b(i, j)$, as described in Equations 4.4, 4.5 and 4.6. The first function is the standard penalization function for the GLS, penalizing long edges. While the other two penalise based on the width. The width of a given edge (i, j) is the distance between the customers i and j and a line (E) which crosses both the depot (D) and the centre of gravity (G) of the route which the edge belongs to, shown in Equation 4.7. These equations compute the penalization cost based on the number of penalisations already made to the edge ($p(i, j)$). Whenever an edge (i, j) has the worst value of $b(.)$, the value of $p(i, j)$ is incremented. More details can be found in the study of [10, 11].

$$b^c(i,j) = \frac{c(i,j)}{1+p(i,j)} \tag{4.4}$$

$$b^w(i,j) = \frac{w(i,j)}{1+p(i,j)} \tag{4.5}$$

$$b^{w,c}(i,j) = \frac{w(i,j)+c(i,j)}{1+p(i,j)} \tag{4.6}$$

$$w(i,j) = max(d_{iE}, d_{jE}) - min(d_{iE}, d_{jE}), E = Line(D,G) \tag{4.7}$$

Their work follows a similar algorithm structure as the one presented in Algorithm 4.7. The paper [10] also adds a new improvement heuristic, namely Relocate-Chain, which moves a customer to another route, even if this breaks the feasibility of the route. In the cases this happens, it is fixed by relocating another customer to make it feasible. This can lead to a chain reaction of infeasible moves until one route is found where the feasibility is restored. Although this move improves the solution quality, it also requires an exponential number of chains and evaluations. The authors [10] avoid that by limiting the number of jumps a move can make. This is an example of how domain knowledge is applied to the problem solution. Our work employs a similar KGLS but as a HH, which reduces some of the domain-specific parameters tuning and allow for a more independent heuristic pool.

4.4.3 GA Hyper-Heuristics

The framework utilises the traditional GA approach where the individuals go through the evolutionary process to optimise the fitness of the individuals. The process is divided into the training and the test phases and is summarised in Figure 4.10.

Training Phase

The training phase consists of a population that goes through the evolutionary process and is initialised at random. At each generation, each

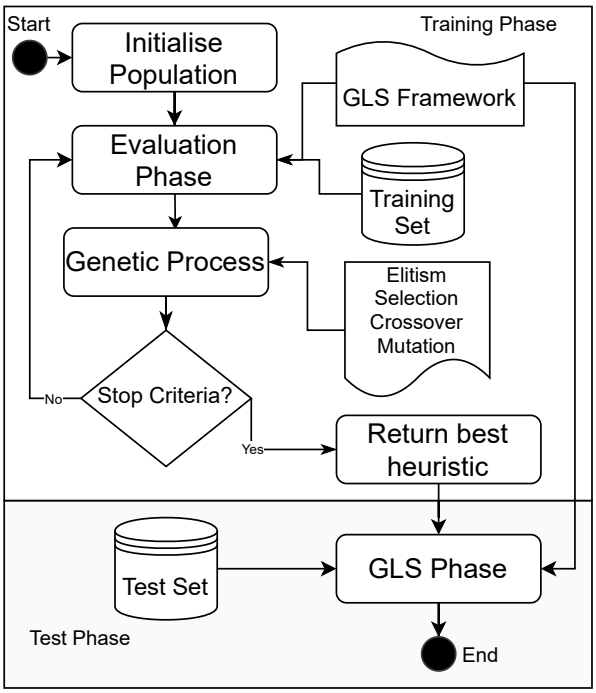


Figure 4.10: Flowchart for the HH framework.

individual is first evaluated. The rest of the evolutionary process follows with an elitism step which saves the best individuals and a crossover and mutation steps that try to improve the population quality for the next generations. These steps are summarized in Algorithm 4.6.

For the *evaluation step*, each individual is evaluated by applying the GLS framework (summarised in Figure 4.11). Then, a normalised function based on how much the solution was improved compared to the initial solution (from CW, which is pre-calculated before execution). This is done to avoid biases towards a larger absolute improvement value. This fitness is calculated by the following equations:

$$N_i^F = \frac{f(x_i)}{CW_i} \tag{4.8}$$

$$fitness^F = \sum_{i=0}^{T} N_i^F \tag{4.9}$$

Algorithm 4.6 Hyper-Heuristic Genetic Algorithm's Training Phase.

procedure GAHH TRAINING(Training Set T, Number of Penalised moves P, Elitist rate L, Crossover rate C, Mutation rate M)
  Calculate CW_i and L_i, i.e. the CW Savings for all Instances $i \in T$ with their average edge cost
  Create a random initial *Population*
  while Stopping criteria not met **do**
    for each Individual $F \in$ Population **do**
      for each Instance $i \in T$ **do**
        $N_i^F \leftarrow$ GLS(Individual,Instance, P)$/CW_i$
        $fitness^F \leftarrow \sum_{i \in T} N_i^F$
    Rank population by fitness
    $H_b \leftarrow$ Rank #1 of population
    Save best individuals according to rate E
    Apply one-point crossover to the population with rate C, following Tournament Selection as the Selection process
    Apply mutation to the population with rate M
  return H_b

In Equation 4.8, we calculate the normalized improvement (N) for one instance i, for individual F, given the VRP solution x_i and the evaluation function f, which is the standard CVRP minimization function, minimizing total distance. The solution after the improvement given by the individual will likely be better than the initial solution constructed by the savings heuristic (CW_i). Therefore, the final value of N_i^F will be between 0 and 1, where the smaller the better. In Equation 4.9, we calculate the fitness of individual F as the sum of the values of N for all the instances in the training set T.

For the *elitism step*, a percentage of the population is saved for the next generations according to the parameter E. The best individuals are added back to the population after the crossover and mutation steps finish, but are still included in the population for the selection process since they can generate better offspring.

As part of the *crossover step*, a Selection process applies a Tournament

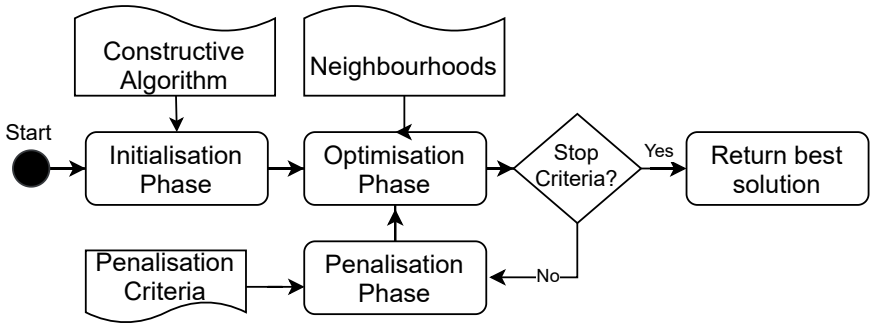


Figure 4.11: Flowchart for the GLS framework.

Selection (TS) to select which pair of individuals to crossover. TS will select random individuals based on the tournament size parameter, and return the fittest individual among them. We apply TS twice to get both parents. Given the pair selected, there is still a probability of mating considering the parameter Crossover rate C. The two parents are replaced by the new offspring in the next generation, keeping the population with the same number of individuals. The parents pair selected is passed to a one-point crossover, which is only applied for the **Optimisation part**, the other parts of the individuals are passed to each child with the same as the parents. The one-point crossover selects a random position from the array and each child will get one of each parents part.

During the *mutation step* all the components of the individuals can be changed. If a given individual passes the mutation rate M, its **Initialisation part** and **Penalisation part** will get a new random initial heuristic or random order (by shuffling the penalisation part), respectively. The Optimisation part will get one of its heuristics modified to a random one.

This model allows the heuristic definition to be controlled by the GA, rather than being manually designed. The elitism guarantees good individuals to be kept in the population. The exchange of information of the crossover operator as well as the randomness of the mutation operator allows for an increase in diversity. The resulting best fit heuristic, i.e. the individual which got a better result across all instances in the training set (on

average), will be the output of the algorithm and will be applied to the test instances [5]. Since larger neighbourhoods might contain better possibilities, it would be expected for the larger limits to obtain better fitness and dominate the population. However, as the running time for each individual is the same, the ones that find better solutions will be more fit, regardless of limits. This way the selection pressure acts in the algorithm and selects the best limits, regardless of the developer's idea of what should be better or faster. If the outcome results in smaller limits, it means that those are enough to find good solutions compared to larger ones[6].

Test Phase

Given the best individual from the previous phase, the unseen instances can be used to measure how effective the trained algorithm is. As the KGLS requires a set of complicated move sets for achieving its maximum capabilities, we re-implemented the algorithm with simpler moves, the same used in our HH, as shown in Section 4.4.1. The KGLS* was also required to do a more fairly comparison to our HH since both use the same components. The test phase simply applies the KGLS* (Algorithm 4.7), with this best individual. The test instances are then evaluated individually by their fitness, just as a regular meta-heuristic.

4.4.4 Experiment Design

In order to avoid unnecessary computation by running the training set for a long amount of time, we performed preliminary experiments with

[5]Regarding time spent searching each neighbourhood, which although is not directly accounted for, there is an underlying factor which will try to optimise it.

[6]Comparing the same operator, if one searches its full neighbourhood for 5 seconds, and it takes the full time to do so, versus another which searches only half its neighbourhood, but does it in half the time, the second version can move from one local optimum from the first search and then to another one for the second run, given the same 5 seconds in total. This can lead to a better overall solution, although it is not guaranteed.

Algorithm 4.7 KGLS* used to evaluate the individuals.

```
1: procedure KGLS EVAL(Individual F, Instance i, Number of penalised moves P)
2:     Find initial solution to i according to F's Initialisation part heuristic
3:     while Stopping criteria not met do
4:         while Local Optimum not reached do
5:             for each LLH ∈ F's Optimisation part do
6:                 Apply LLH with its limits to the current solution
7:         Change penalisation function b to the next one in F's Penalisation part j
8:         while Number of penalised moves ≤ P do
9:             Penalise current solution with current penalise function b_j
10:            Select penalised edge e
11:            for each LLH ∈ F's Optimisation part do
12:                Apply LLH with its limits to edge e considering penalised edges
13:            if A move was made then
14:                Increment number of penalised moves
15:    return value of best solution
```

the KGLS* adaptation to determine fair parameters for our EHH. Those experiments consisted of running the KGLS* for all instances for the same amount of time (10 minutes) and analysing their convergence speed and curves. Figure 4.12 shows the quality of the solution over time for the test set. The graph shows how much the solution has improved from the starting solution, relatively. As can be seen from the figure, most of the improvement happens within the first seconds of execution. Therefore, we determined that the training time limit for each individual in our hyper-heuristic would be of 30 seconds, which should be enough for finding the best improvement application of the heuristics. A similar trend was observed for the training set.

The experiment was designed to verify whether our EHH can find a better heuristic configuration than the manually designed KGLS. In this experiment, the EHH is compared to how good the final solution is, given the same amount of time. This aims to show whether our approach can find a better final solution compared to the KGLS*. This experiment also

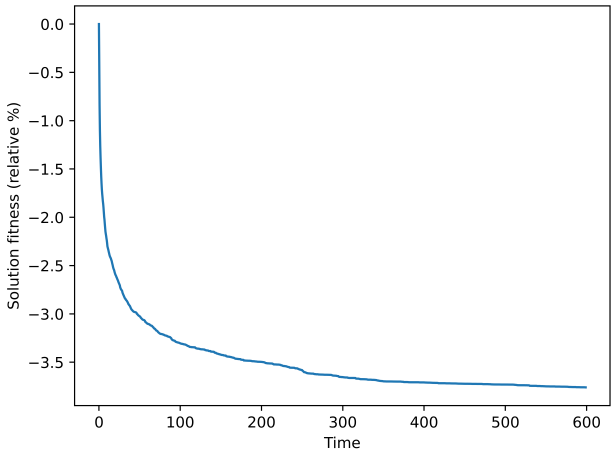


Figure 4.12: Improvement over time for the KGLS* implementation considering the test instance set.

compares the final result to the original KGLS [9] and to the Best-Known-Solution (BKS)[7]. The results for the KGLS were collected by running the available online tool the authors provided[8].

Finally, in order to have the experiments running faster, and to take advantage of modern computers with multi-threaded CPUs, we added a parallel evaluation of the individuals in a given generation. Since each individual evaluation is independent of one another, we can safely do this and save a significant amount of time in the training phase. Since we are running the training set with the instances which have a smaller number of customers, memory is also easier to be handled, as it will not require as much as the larger instances.

4.4.5 Training and Testing

The experiments were realized with the VRPLib dataset [183], which contains 100 instances varying in size (from 100 to 1000 customers), in the

[7]Retrieved from the dataset website: http://vrp.atd-lab.inf.puc-rio.br/index.php/en/

[8]Available on the website: https://antor.uantwerpen.be/routingsolver/

distribution of customers (randomly distributed, clustered, mixed), in customers demand (from 1 to 100, with different combinations of sizes), vehicle capacity (varies according to the demand for the instance) and depot location (central, eccentric and random). These different instances try to simulate several real-world scenarios in which the VRP can be applicable. A good solution method would perform well in all scenarios, even though that is hard to achieve since in general different heuristics work better under different situations.

The training set used was composed of all instances smaller than 200 customers, containing 21 instances (from 100 to 199). While the rest (79 instances) were used as the test set. We assume the sample of the 21 instances have enough variety for the algorithm to learn properly and apply the knowledge efficiently for the larger instances. The instances with fewer customers are also easier to compute due to the reduced search space and are able to find better solutions within a small amount of time. And although this does not necessarily translate to better quality for larger instances, it is a good indication according to our preliminary experiments (as shown in Figure 4.12).

4.4.6 Parameters

One of the main parameters of the KGLS is the number of penalised moves before moving to the optimisation phase. The authors of the original work have tried with 10, 100 and 1000, for different configurations of their KGLS (by comparing with different components). Although their final performance was better with 100, the 10 was actually better for the simple version of the KGLS with less advanced components. We compared this value and confirmed that 10 is better for our version too, and the value was used for our EHH.

For our EHH, the parameters are as shown in Table 4.12. The population size and number of generations were determined by a few pre-

liminary experiments, finding that larger numbers would take too long to evaluate all 21 training instances. The bounds for the optimisation part of the chromosome are wide enough to allow for the evolution to determine the better values.

Table 4.12: The GA Hyper-Heuristics parameters.

Parameter	Value
Population size	20
No. of Generations	50
Cross. Rate	0.8
Mut. Rate	0.1
Elite Rate	0.1
Lower bound (for limit and decay)	0.1
Upper bound (for limit and decay)	0.9
Number of runs	30

All tests were conducted in a Intel®Core™i7-8700 @ 3.2GHz and 15GB available memory, using half of the 12 threads for the parallel evaluation, and the implementation was done with C++ version 17.

4.4.7 Results and Discussions

This section shows the results obtained from the experiments of this given method. The results compared are for the test set only. We comment on the results based on the average case, best case and the evolved individuals over the 30 runs. The rank-sum Wilcoxon statistical test was performed to verify the significance of the results (p-value $= 0.05$).

Average Case: For the average case, we can observe that the EHH proposed performs relatively poor, as shown in Table 4.13. On average, the EHH only beats (with significance, on the table in boldface) the KGLS* in 11 out of the 79 instances and drawing in 12 of them (no significant advantage or disadvantage). The results are even more significant if compared to the original KGLS, not beating it in any case. This, however, is to be

expected since they apply some complex neighbourhood moves (such as the LK heuristic and the Relocate Chain).

Table 4.13: Results compared to BKS. In bold are the best between KGLS* and EHH, while underlined are the overall best.

Instance	KGLS to BKS	KGLS* to BKS	EHH Avg. to BKS	EHH Best to BKS
X-n200-k36	0.29%	2.28%	**1.37%**± 0.57%	**0.84%**
X-n204-k19	0.52%	1.05%	1.09% ± 0.26%	**0.35%**
X-n209-k16	0.25%	1.69%	**1.22%**± 0.48%	**0.62%**
X-n214-k11	0.67%	**2.46%**	3.29%± 1.01%	**1.75%**
X-n219-k73	0.07%	0.15%	0.20%± 0.07%	**0.07%**
X-n223-k34	0.59%	1.79%	**1.57%**± 0.25%	**1.26%**
X-n228-k23	0.37%	**1.37%**	2.34%± 0.52%	**1.13%**
X-n233-k16	0.54%	**1.12%**	1.89%± 0.46%	**0.93%**
X-n237-k14	0.24%	0.82%	**0.65%**± 0.39%	**0.29%**
X-n242-k48	0.47%	1.23%	1.19%± 0.23%	**0.72%**
X-n247-k47	1.11%	2.60%	**2.07%**± 0.62%	**1.34%**
X-n251-k28	0.60%	1.64%	**1.33%**± 0.20%	**1.01%**
X-n256-k16	0.05%	**0.80%**	1.52%± 0.54%	**0.75%**
X-n261-k13	0.43%	**2.00%**	2.57%± 0.64%	**1.41%**
X-n266-k58	0.64%	1.31%	1.33%± 0.19%	**0.91%**
X-n270-k35	0.45%	**0.95%**	1.27%± 0.21%	**0.95%**
X-n275-k28	0.16%	**0.97%**	1.19%± 0.22%	**0.81%**
X-n280-k17	0.56%	**2.28%**	3.74%± 0.43%	2.96%
X-n284-k15	0.80%	**2.31%**	2.84%± 0.42%	**2.25%**
X-n289-k60	0.86%	1.56%	1.61%± 0.26%	**1.16%**
X-n294-k50	0.41%	1.65%	**1.43%**± 0.22%	**0.97%**
X-n298-k31	0.41%	**1.34%**	2.09%± 0.32%	1.55%
X-n303-k21	0.61%	**1.72%**	2.71%± 0.46%	2.06%
X-n308-k13	1.17%	**2.39%**	3.77%± 0.95%	**1.64%**
X-n313-k71	0.93%	1.49%	1.57%± 0.22%	**1.19%**
X-n317-k53	0.07%	0.41%	0.53%± 0.21%	**0.28%**
X-n322-k28	0.58%	**1.94%**	2.26%± 0.41%	**1.53%**
X-n327-k20	0.40%	**1.33%**	2.10%± 0.55%	1.43%
X-n331-k15	0.24%	**1.28%**	1.90%± 0.48%	**1.14%**

Table 4.13: Results compared to BKS. In bold are the best between KGLS* and EHH, while underlined are the overall best (cont.).

Instance	KGLS to BKS	KGLS* to BKS	EHH Avg. to BKS	EHH Best to BKS
X-n336-k84	1.03%	**1.66%**	1.82%± 0.34%	**1.42%**
X-n344-k43	0.71%	**1.57%**	1.92%± 0.30%	**1.43%**
X-n351-k40	0.82%	2.58%	**2.15%**± 0.29%	**1.52%**
X-n359-k29	0.93%	**1.52%**	2.04%± 0.37%	**1.42%**
X-n367-k17	0.51%	**2.04%**	2.65%± 0.64%	**1.43%**
X-n376-k94	0.09%	**0.32%**	0.38%± 0.07%	**0.24%**
X-n384-k52	0.44%	1.39%	**1.34%**± 0.26%	**0.89%**
X-n393-k38	0.53%	1.93%	1.96%± 0.33%	**1.42%**
X-n401-k29	0.54%	**1.05%**	1.19%± 0.28%	**0.81%**
X-n411-k19	1.89%	**2.70%**	6.11%± 0.58%	**4.81%**
X-n420-k130	0.56%	1.35%	**1.24%**± 0.41%	**0.71%**
X-n429-k61	0.43%	**1.58%**	2.09%± 0.38%	**1.35%**
X-n439-k37	0.39%	**0.93%**	1.42%± 0.22%	1.10%
X-n449-k29	0.75%	2.55%	2.75%± 0.51%	**1.84%**
X-n459-k26	0.20%	**1.80%**	3.48%± 0.54%	2.86%
X-n469-k138	0.66%	**1.20%**	2.10%± 0.28%	**1.44%**
X-n480-k70	0.38%	**1.27%**	1.78%± 0.27%	1.32%
X-n491-k59	0.75%	**1.78%**	2.00%± 0.25%	**1.60%**
X-n502-k39	0.11%	0.57%	0.56%± 0.12%	**0.44%**
X-n513-k21	0.44%	**2.16%**	6.51%± 1.28%	4.05%
X-n524-k137	1.77%	**2.66%**	3.55%± 0.49%	**2.50%**
X-n536-k96	0.81%	**1.55%**	2.19%± 0.37%	1.60%
X-n548-k50	0.25%	1.00%	1.03%± 0.16%	**0.71%**
X-n561-k42	0.63%	**1.71%**	3.16%± 0.61%	1.96%
X-n573-k30	0.19%	**1.39%**	2.21%± 0.23%	1.82%
X-n586-k159	0.53%	**1.19%**	2.18%± 0.28%	1.63%
X-n599-k92	0.54%	**1.22%**	1.67%± 0.15%	1.39%
X-n613-k62	0.72%	**2.27%**	2.93%± 0.27%	2.43%
X-n627-k43	0.31%	**1.78%**	2.45%± 0.26%	1.99%
X-n641-k35	0.39%	**1.90%**	3.37%± 0.38%	2.68%
X-n655-k131	0.19%	0.49%	0.53%± 0.08%	**0.40%**

Table 4.13: Results compared to BKS. In bold are the best between KGLS* and EHH, while underlined are the overall best (cont.).

Instance	KGLS to BKS	KGLS* to BKS	EHH Avg. to BKS	EHH Best to BKS
X-n670-k126	3.28%	**5.33%**	6.28%± 0.32%	5.83%
X-n685-k75	0.70%	**1.72%**	2.90%± 0.22%	2.37%
X-n701-k44	0.22%	**1.55%**	3.12%± 0.17%	2.72%
X-n716-k35	0.69%	**2.60%**	4.42%± 0.19%	3.96%
X-n733-k159	0.61%	**1.39%**	1.67%± 0.14%	**1.33%**
X-n749-k98	0.58%	1.69%	**1.56%**± 0.12%	**1.36%**
X-n766-k71	1.04%	**2.84%**	3.69%± 0.12%	3.47%
X-n783-k48	0.49%	**1.60%**	3.31%± 0.25%	2.72%
X-n801-k40	0.07%	**0.96%**	2.47%± 0.25%	1.82%
X-n819-k171	0.61%	**1.16%**	3.00%± 0.29%	2.51%
X-n837-k142	0.46%	**1.26%**	2.21%± 0.17%	1.83%
X-n856-k95	0.34%	**0.83%**	1.53%± 0.21%	1.28%
X-n876-k59	0.42%	**1.67%**	1.86%± 0.12%	1.71%
X-n895-k37	0.20%	**2.54%**	6.05%± 0.43%	4.88%
X-n916-k207	0.43%	**0.91%**	2.56%± 0.25%	2.10%
X-n936-k151	3.40%	**6.22%**	8.03%± 0.41%	7.32%
X-n957-k87	0.47%	**0.84%**	1.67%± 0.26%	1.49%
X-n979-k58	0.55%	**1.23%**	2.53%± 0.20%	2.16%
X-n1001-k43	0.40%	**2.85%**	5.10%± 0.19%	4.79%
Average	**0.61%**	**1.67%**	2.37%	1.77%

Best Case: The best case, however, shows encouraging results. Also in Table 4.13, the best EHH gap to the BKS is shown. The best EHH beats the KGLS* in 46 of the 79 instances, and even beating the original KGLS for one instance. However, even considering this case, the overall average is still shy of the KGLS*.

By analysing the results, we can see that most of the EHH best results are on the smaller side (less than 600 customers), losing in most of the larger instances. This could indicate that the heuristic configuration that performs well for the training set is not generalisable for larger instances.

When looking at the best-evolved heuristics, shown in Figure 4.14, and how they perform over time, we see that they present very similar behaviour. Showing that our EHH is robust to the randomness of the initial population, being all individuals within the $2 \sim 2.5\%$ relative improvement range, with only one outlier.

Without surprise, all best heuristics had the Savings algorithm as the initialisation method, since it provides a much better initial solution. The order of the penalisation operators appears to be less relevant, with most methods presenting the same order as the original KGLS* (sometimes shifted). Figure 4.13 shows an example of one of the best heuristics evolved.

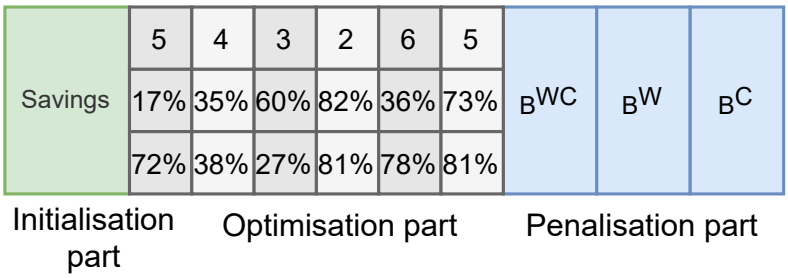


Figure 4.13: Example of heuristic evolved individual.

4.4.8 Summary

In this section we present our third hyper-heuristic with a novel chromosome which incorporates the three main steps of the KGLS — initialisation, optimisation and penalisation. The training process tries to optimise the order for which these components perform better across the different instances. We also introduce some chromosome-specific evolutionary operators in the new crossover and mutation. On one side, the results show a competitive performance even with simpler LLH. On the other side, however, it showed that the new HH only perform well for the instances with up to 500 customers. One possible reason for the unsatisfactory performance on the larger instances is the evaluation of each training instance

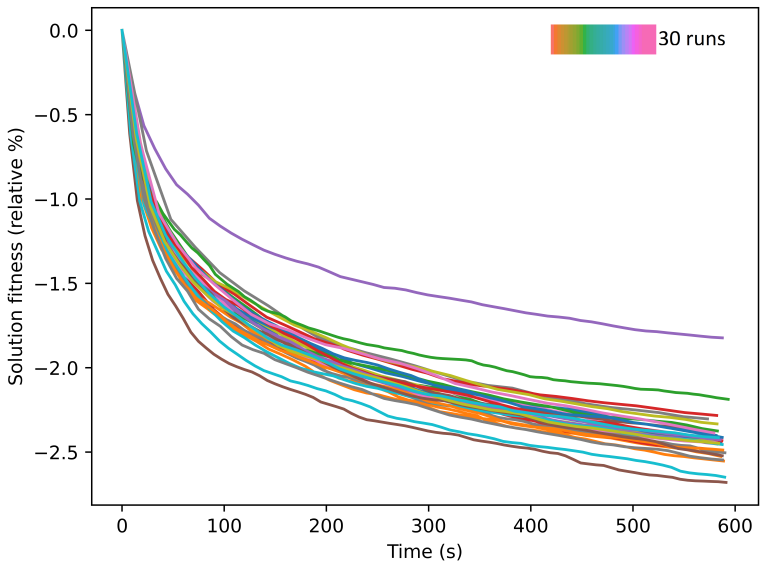


Figure 4.14: Performance of the final heuristic over time, for all 30 runs.

bound by a fixed time. Therefore, the EHH would only learn to efficiently solve the instances with a similar number of customers to those of the training set.

4.5 Chapter Summary

In this chapter we present the automatic modelling of the improvement step through the use of selection HHs. We present 3 different methods for doing so, each with its unique modifications. The first one applies a novel adaptive clustering technique which reduces the search space progressively while looking for the best order of neighbourhoods to be used. Achieving comparatively good results to other heuristics methods, while also being more efficient.

The second method introduces a new dual-layer chromosome, where each allele represents a neighbourhood operator and its search size. The use of this novel chromosome is tested in simple local search frameworks, but is able to show its efficiency when compared to a fixed size strategy.

The third method is a bit more complex, incorporating more steps of the local search framework. For this method we introduce another novel chromosome with a separate part for the initialisation heuristic and another part for the penalisation steps of the KGLS, as well as an additional layer to reduce searches between far routes.

This chapter showed that it is possible to build efficient selection hyper-heuristics, but they show an impact in the effectiveness of the methods, although this is not as significant for several cases.

4.5.1 Consideration on Scalability and Computational time

Three methods have been presented in this chapter. We comment their relationship to the scalability and computational time individually.

For the first method, the creation of clusters scales with the number of customers and the number of clusters being created. Following that, the scale of the instances will also affect the local search performance, as each neighbourhood considered will have more neighbours to evaluate. The computational time for this approach, therefore, can be quite high, ranging from minutes to hours, even with instances of less than 1000 customers.

The second approach has very similar concerns compared to the previous method. Although the effects on scalability and computational time are lessened down because no clusters are considered, the neighbouring solutions searched are still relatively high. This is especially true with some given configurations of the individual at the end of the optimisation. For example, a HH solution that has an average of 30% in its second layer, will have a very different number of solutions searched depending on the problem. For an instance with 250 customers it would search 75 neighbours, while an instance with 1000 would search for 300. Again, it can take several hours for a single run of the evolved heuristic.

The last approach is a bit less sensitive to the scalability as it adds another limit to the search space with the third layer. This layer helps to scale

down the number of neighbouring solutions by limiting how far the local search can explore. When combining the KGLS framework on top of these layers, we get to keep the computational time to a level that is comparable to the original KGLS.

Chapter 5

Learning to Guide Solution Search Space

In this chapter we introduce our strategy for adjusting the solution search space size in a more fine tuned and independent way.

5.1 Introduction

In a local search-based metaheuristic, the *neighbourhood pruning* strategy is widely used to improve search efficiency, especially for large scale problems, without losing much effectiveness. Specifically, the neighbourhood search adopts one or more *move operators*. Each move operator $\mathtt{op}$ leads to a corresponding neighbourhood, i.e., $\mathcal{N}_{\mathtt{op}}(S) := \{S' \mid S' = \mathtt{op}(S)\}$. Each neighbour $S' \in \mathcal{N}_{\mathtt{op}}(S)$ of S is a slight modification of S by the operator. The local search typically selects a better neighbour $S^* \in \mathcal{N}_{\mathtt{op}}(S)$, where $f(S^*) < f(S)$ (for minimisation problem), until a local optimum is found [1,140]. However, the neighbourhood size increases rapidly with the increase of problem size. For example, the neighbourhood size of the exchange operator is $O(n^2)$, where n is the number of customers. Neighbourhood pruning can limit the search within a much smaller subset of the neighbourhood $\overline{\mathcal{N}}_{\mathtt{op}}(S) \subset \mathcal{N}_{\mathtt{op}}(S)$ to greatly reduce the search space with-

out losing promising neighbours. The intuition of neighbourhood pruning is that some neighbours are obviously unpromising (e.g., exchanging two customers where their new positions are both distant from their adjacent customers in the resultant solution), and thus do not need to be examined in the neighbourhood. The neighbourhood pruning techniques attempt to identify and remove such obviously unpromising neighbours to obtain a much smaller neighbourhood. A common strategy is to use the closeness between the moved customers and their adjacent customers. For instance, a neighbouring solution is in the pruned neighbourhood only if the distance between each moved customer and its adjacent customers is lower than a predefined threshold.

However, it is non-trivial and challenging to design an effective neighbourhood pruning scheme for solving the LSVRP, as the best neighbourhood size depends on various factors such as the characteristics (e.g., graph topology, demand distribution) of the LSVRP instance and the move operator considered. Existing methods (such as [9]) design neighbourhood pruning strategies that set the threshold on the distances based on the graph topology (e.g., the distance threshold for a customer is set to its distance to the Kth closest customer to it in the graph). Nevertheless, the parameter K is instance and operator dependent, and thus it is challenging to find its best value.

There are two main approaches to finding the best neighbourhood pruning parameters: (1) offline parameter tuning and (2) online parameter adaptation. Offline parameter tuning empirically compares the performance of different parameters on a set of (training) instances and selects the parameter value with the best performance. The offline strategy was mainly applied in Chapter 4. However, as shown, the generalisation could be an issue, as it is difficult to ensure that the training instances are representative enough of all the different kinds of LSVRP instances (graph topology, demand distribution, number of vehicles, etc.) in the real world. In addition, offline parameter tuning typically suggests a fixed parameter

value, while different instances in practice might require different best parameter values. Online parameter adaptation, on the other hand, is more flexible, as it can adapt to every single instance, learning the best parameter values during the search process. In this chapter, we focus on online parameter adaptation.

5.1.1 Chapter Goals

The overall goal of this chapter is to develop an online neighbourhood pruning adaption strategy for solving LSVRP more effectively and efficiently. The specific research goals are:

1. To design a new heuristic to adapt the neighbourhood search size, i.e., the distance threshold during the search process. This heuristic increases or decreases the threshold based on the gap between the current solution and the previous one. Intuitively, it retains or decreases the threshold if there is an improvement in the previous iteration while increasing the threshold if the search is currently stuck into a local optimum.

2. To embed the neighbour size adaptation heuristic into the KGLS framework, which is a recent and state-of-the-art algorithm for LSVRP.

3. Verify the effectiveness of the KGLS with neighbourhood size adaptation on a wide range of LSVRP instances and analyse the behaviour of the proposed heuristic.

5.1.2 Chapter Organisation

The rest of this chapter is organised as follows: the proposed KGLS with the neighbourhood size adaption heuristic is elaborated in Section 5.2. The experiment design is presented in Section 5.3. In section 5.4, the main results are shown followed by Section 5.5 presenting further discussions and analysis. Finally, the chapter summary is presented in Section 5.6.

5.2 Proposed Method

The newly proposed approach adopts the KGLS framework [10], which is a recent state-of-the-art guided local search algorithm for LSVRP. The only difference is that our new approach incorporates a new heuristic to adapt the neighbourhood size during the search. In the following, we will first describe the overall algorithm and then describe the new neighbourhood size adaptive heuristic.

5.2.1 Overall Framework

The KGLS is again utilised as our baseline local search-based metaheuristic. Algorithm 5.1 shows the overall framework of the proposed method. First, it initialises a solution using the Savings* heuristic (originally from [37], adapted by [9]), and set the initial neighbourhood sizes to $\Delta_{CE} = \Delta_{RC} = 30$ according to [10]. Then, during the search process, it generates new solutions using three distinct move operators. For intra-route, the Lin-Kernighan Heuristic (LKH) is used. For inter-route, the Cross Exchange (CE) and Relocation Chain (RC) are used. At each iteration, it searches for a local optimum using the three operators. Then, it calculates the penalisation term, changes the objective function through penalisation, and continue the search under the new objective function to jump out of the local optimum. Note that in Algorithm 5.1, the neighbourhood of CE and RC are pruned by the parameters Δ_{CE} and Δ_{RC}. At the end of each iteration, the neighbourhood sizes of CE and RC are adapted by the newly developed heuristic. This is the main difference from the original KGLS (lines 17–19 as highlighted in bold).

The details of the CE and RC operators under the pruned neighbourhood are given in Algorithm 5.2, with a unified notation $\mathtt{op}(\cdot)$. At first, for each node, the subset of the Δ closest neighbours are obtained by $\mathtt{Closest}(V \setminus v_i, v_i, \Delta)$. To this end, we sort all the neighbours $v' \in V \setminus v_i$ in the increasing order of their distance to v_i, and then select the top Δ ele-

Algorithm 5.1 Newly Proposed Method

```
1: procedure KGLS(instance i)
2:     Generate initial solution S using Savings heuristic
3:     Set Δ_CE = Δ_RC = 30
4:     while Stopping criteria not met do Start optimisation phase
5:         while Local optimum not reached do
6:             S' ← CE(S, Δ_CE)
7:             S'' ← RC(S', Δ_RC)
8:             S ← LKH(S'')
9:         Change penalisation function   Start penalisation phase
10:        while Number of penalised moves not reached do
11:            S' ← Penalise(S)
12:            Select penalised edge from S'
13:            Change objective function
14:            S'' ← CE(S', Δ_CE), S ← RC(S'', Δ_RC)
15:            if A move was made then
16:                Increment number of penalised moves
           S ← LKH(S)
17:        Collect the best neighbour indexes Ω_CE, Ω_RC
18:        Update Δ_CE ← AdaptNS(Δ_CE, S, S', Ω_CE)
19:        Update Δ_RC ← AdaptNS(Δ_RC, S, S', Ω_RC)
20:    return best solution
```

ments in the sorted list. Note that the subset for each node can be obtained as a preprocessing stage to avoid duplicate computation. For each candidate move (neighbour) that adds new edges E_{new}, if, for any new edge, an end-node is outside the closest neighbour subset of the other end-node, then the edge violates the neighbourhood size limit, and the candidate move is skipped. For CE, each exchange contains four new edges, while for RC, each relocation can lead to up to eight additional edges.

5.2.2 Neighbourhood Size Adaption Heuristic

The newly proposed neighbourhood size adaption heuristic changes the neighbourhood size based on the feedback of the current iteration (line

Algorithm 5.2 $S' \leftarrow \mathtt{op}(S, \Delta)$

procedure OPERATOR(solution S, neighbourhood limit Δ) $S' \leftarrow S$, $sav^* \leftarrow 0$
  for each $v_i \in V$ **do**
    $\mathcal{V}_i \leftarrow \mathtt{Closest}(V \setminus v_i, v_i, \Delta)$
  for each candidate move that adds new edges E_{new} **do**
    if $\exists e \in E_{new}$, $head(e) \notin \mathcal{V}_{tail(e)}$ **then**
      continue
    Calculate the savings by removing the old edges and adding the new edges
    if $sav > sav^*$ **then**
      $S' \leftarrow$ apply the move on S
      $sav^* \leftarrow sav$
  return S'

17 in Algorithm 5.1). The feedback is represented as the best neighbour indexes Ω_{CE} and Ω_{RC}. Specifically, for each node, its neighbours are indexed in the increasing order of their distance to the node (i.e., the closest neighbour has index 0). Then, for each move from S to S', for each moved node, we record the index of its new adjacent node in S' in its neighbour list and add this index into the corresponding set Ω. Since the neighbour size depends on the operator, we keep two sets of indexes, namely Ω_{CE} and Ω_{RC}. The depot is not considered for this, for obvious reasons.

Next, the neighbourhood size adaption is done based on the feedback Ω_{CE} and Ω_{RC} as well as the start solution S and end solution S' of the current iteration. Given a move operator (e.g., CE or RC), the neighbourhood size adaption heuristic is shown in Algorithm 5.3. First, it obtains the maximal index stored in Ω, i.e., the index of the best neighbour in the worst case among the moves in this iteration. Then, it adjusts the neighbourhood size Δ as follows. If the new solution S' is better than S, then an improvement is found in the iteration. In this case, if the maximal index is larger than $\Delta - \delta$ (δ is a parameter), then it means that in the worst case the best neighbour is close to the current neighbourhood size limit. Therefore, there is a risk that the best neighbour in the next iteration will exceed the current limit and cannot be found. To reduce such risk, we increase the

neighbourhood size by δ with a predefined probability of θ. Otherwise, even in the worst case, the best neighbour is in a safe region of the limit, and there tends to be a low risk that the best neighbour in the next iteration will exceed the limit. Furthermore, it suggests that the current limit might be too large, and we can safely decrease it without losing the best neighbour. Thus, we decrease Δ by δ with the probability of θ. On the other hand, if there is no improvement found in the iteration, then we increase Δ by δ with a probability of θ. This is done to increase the chances of jumping out of the local optimum that was just met (as there was no improvement). Note that we always have a certain probability to retain the neighbourhood size unchanged. This is to avoid changing the neighbourhood size too rapidly, leading to fluctuating and non-smooth behaviour.

Algorithm 5.3 `AdaptNS`(Δ, S, S', Ω)

procedure `AdaptNS`(Δ, S, S', Ω)(Neighbourhood limit Δ, Old solution S, New solution S', Best neighbour indexes Ω)
  $\Delta' \leftarrow \Delta$
  $\bar{i} \leftarrow \max\{i \mid i \in \Omega\}$
  if S' is better than S **then**
    if $\bar{i} > \Delta - \delta$ **then**
      Change $\Delta' \leftarrow \Delta + \delta$ with probability θ
    else
      Change $\Delta' \leftarrow \Delta - \delta$ with probability θ
  else
    Change $\Delta' \leftarrow \Delta + \delta$ with probability θ
  return Δ'

To verify the idea of the adaptive neighbourhood size heuristic, we plot the maximal index of the best neighbours for CE and RC over 50 iterations of the original KGLS (with a fixed neighbourhood size of 30) in Figures 5.1 and 5.2. In both figures, we can see that most of the time the best neighbour has a much smaller index than 30, which means that we can reduce the neighbourhood size without losing them. However, a few neighbours have indexes close to 30, meaning that we will lose them if setting a smaller

fixed neighbourhood size. In addition, we can see some degree of auto-correlation (i.e., a large/small index followed by a relatively large/small index). This demonstrates the potential to adjust the neighbourhood size based on the feedback of the current iteration.

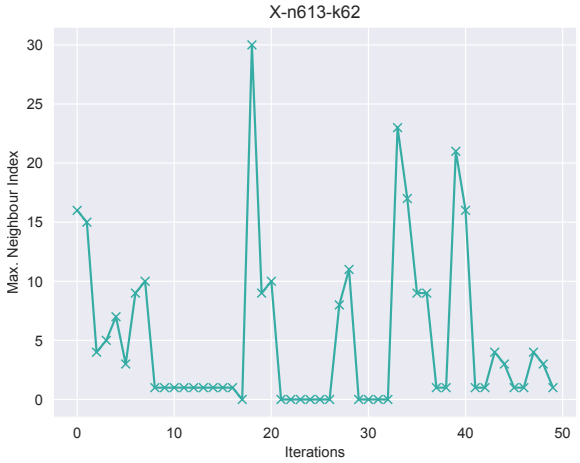


Figure 5.1: The index of the best neighbour for the *Cross-Exchange* in 50 iterations of KGLS.

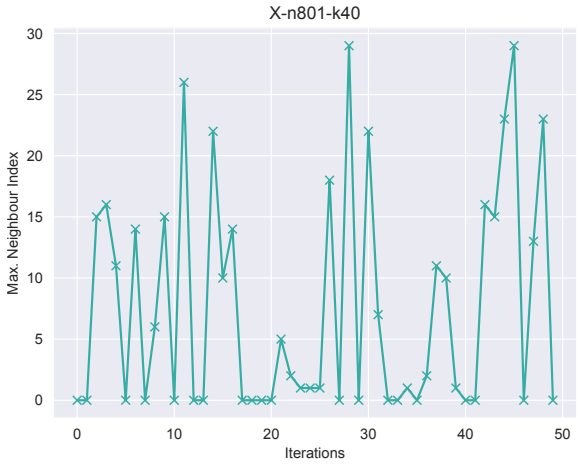


Figure 5.2: The index of the best neighbour for the *Relocation Chain* in 50 iterations of KGLS.

5.3 Experiment Design

The datasets utilised in the experiments are two of the most recently published. The first "X" dataset [183] contains 100 instances ranging from 100 to 1000 customers. They present several customer distribution types, including Random (R), Clustered (C) and Random-Clustered (RC), which is a mix between both types. The dataset also has a diverse customer demand, vehicle capacity, and position of the depot (in a corner, at the centre or a random location). All these characteristics allow for simulating diverse real-world scenarios. Although some of the instances are not classified as large-scale (having less than 200 customers), they are still kept for a fairer comparison to other methods.

The second dataset is the very-large-scale instances introduced in [9]. The number of customers ranges from 3000 to 30000 over 10 instances. These instances are based on a real-life scenario and, therefore, have less diversity regarding the distribution. However, they are suitable to validate the effectiveness of methods in the real world.

The instances were divided in three categories for easier discussions in the next section, a similar division is used by [6]: **medium**, for the "X" instances with sizes up to 350; **large**, for the remaining of the "X" instances (from 350 up to 1000); and **very-large**, for the instances from [9], which have at least 3000 customers.

The KGLS is used as the baseline comparison algorithm and utilised its default parameters as in [10]. For the new parameters for neighbourhood size adaption, θ is set to 75% and δ is set to 5 after some preliminary experiments. In addition, we set an upper limit of 30 and a lower limit of 10 for the neighbourhood size during the search. For stopping criteria, we set a maximal time limit of 10 minutes for all the compared algorithms for a fair comparison. The parameter setting is shown in Table 5.1.

Table 5.1: Parameter settings

Parameter	Values
θ	75%
δ	5
Neighbourhood size upper limit	30
Neighbourhood size lower limit	10
Time limit	10 minutes

5.4 Results and Discussions

The expectation with the adaptive neighbourhood size adaption heuristic is to find solutions of similar quality than the original fixed neighbourhood size given the same number of iterations (or at least not worse than). In addition, as the reduced search space would allow for a larger number of iterations to happen in the same time frame, this could result in finding new and better solutions.

Tables 5.2–5.4 shows the results of the compared algorithms and Best-Known Solution (BKS)[1] on the medium, large and very large instances. For each instance, KGLS has a single number, since it is deterministic. The proposed algorithm is run 30 times independently, and the best, mean and standard deviation are recorded. We also compare with the results of KGLS by the Wilcoxon rank-sum test with a significance level of 0.05. The notation (+)/(-)/(=) indicates that the proposed algorithm performs statistically better/worse/comparable with the original KGLS.

Table 5.2: Results of 30 runs on the *Medium* instances.

	BKS	KGLS	Proposed Algorithm		
Instance	Total Cost	Total Cost	Best	Average	Std. Dev.
X-n101-k25	27591	27615	27591	**27604.7**	11.6 (+)
X-n106-k14	26362	**26401**	26387	26413.6	10.9 (-)

[1]The BKS for both the "X" instances and the very-large-scale from [9] were collected from the CVRPLib website: `vrp.atd-lab.inf.puc-rio.br/index.php/en/`

Table 5.2: Results of 30 runs on the *Medium* instances (cont.).

	BKS	KGLS	Proposed Algorithm		
Instance	Total Cost	Total Cost	Best	Average	Std. Dev.
X-n110-k13	14971	14971	14971	14971.0	0.0 (=)
X-n115-k10	12747	12747	12747	12747.0	0.0 (=)
X-n120-k6	13332	13332	13332	13332.0	0.2 (=)
X-n125-k30	55539	56140	55647	**56091.9**	113.4 (+)
X-n129-k18	28940	28973	28956	**28966.0**	6.8 (+)
X-n134-k13	10916	**10916**	10917	10935.0	7.9 (-)
X-n139-k10	13590	13590	13590	13590.0	0.0 (=)
X-n143-k7	15700	**15726**	15726	15729.1	2.7 (-)
X-n148-k46	43448	43539	43494	43540.4	31.1 (=)
X-n153-k22	21220	21389	21377	21698.9	291.7 (=)
X-n157-k13	16876	16876	16876	16876.1	0.5 (=)
X-n162-k11	14138	14147	14138	14146.6	1.7 (=)
X-n167-k10	20557	20589	20557	20578.1	16.7 (=)
X-n172-k51	45607	45684	45623	**45664.0**	23.9 (+)
X-n176-k26	47812	48786	47880	**48508.4**	347.8 (+)
X-n181-k23	25569	25615	25572	**25596.2**	7.1 (+)
X-n186-k15	24145	24184	24145	**24169.3**	14.8 (+)
X-n190-k8	16980	17036	17005	**17029.3**	19.6 (+)
X-n195-k51	44225	44453	44298	**44417.1**	38.2 (+)
X-n200-k36	58578	58811	58742	**58802.4**	35.8 (+)
X-n204-k19	19565	19666	19583	19655.7	44.2 (=)
X-n209-k16	30656	30733	30686	**30715.0**	14.0 (+)
X-n214-k11	10856	**10919**	10913	10951.2	20.1 (-)
X-n219-k73	117595	117674	117627	**117669.2**	19.6 (+)
X-n223-k34	40437	40708	40654	40705.6	22.2 (=)
X-n228-k23	25742	**25838**	25817	25874.1	33.3 (-)
X-n233-k16	19230	19333	19272	**19331.0**	24.2 (+)
X-n237-k14	27042	27074	27050	27101.8	20.5 (-)
X-n242-k48	82751	83265	83044	**83202.6**	55.9 (+)
X-n247-k47	37274	37691	37481	**37673.0**	36.2 (+)
X-n251-k28	38684	38916	38800	**38880.5**	39.2 (+)
X-n256-k16	18839	18890	18888	18891.7	4.9 (=)
X-n261-k13	26558	26720	26666	26713.5	34.7 (=)

Table 5.2: Results of 30 runs on the *Medium* instances (cont.).

	BKS	KGLS	Proposed Algorithm		
Instance	Total Cost	Total Cost	Best	Average	Std. Dev.
X-n266-k58	75478	75964	75766	**75936.5**	54.6 (+)
X-n270-k35	35291	**35414**	35399	35455.1	20.5 (-)
X-n275-k28	21245	**21280**	21253	21298.8	20.9 (-)
X-n280-k17	33503	**33661**	33609	33678.4	33.9 (-)
X-n284-k15	20215	20388	20318	20382.0	29.8 (=)
X-n289-k60	95151	96023	95735	**95917.3**	75.1 (+)
X-n294-k50	47161	**47419**	47395	47466.1	34.8 (-)
X-n298-k31	34231	34395	34348	**34380.1**	14.3 (+)
X-n303-k21	21736	21931	21819	**21901.2**	28.2 (+)
X-n308-k13	25859	**26050**	25969	26069.0	69.6 (-)
X-n313-k71	94043	**94845**	94748	94874.3	68.5 (-)
X-n317-k53	78355	78409	78382	78413.3	11.9 (=)
X-n322-k28	29834	30038	29955	**30021.4**	26.1 (+)
X-n327-k20	27532	27667	27611	27662.4	28.7 (=)
X-n331-k15	31102	31178	31106	31193.5	61.9 (=)
X-n336-k84	139111	**140668**	140589	140767.1	89.8 (-)
X-n344-k43	42050	42398	42225	**42331.9**	59.9 (+)

Table 5.3: Results of 30 runs on the *large* instances.

	BKS	KGLS	Proposed Algorithm		
Instance	Total Cost	Total Cost	Best	Average	Std. Dev.
X-n351-k40	25896	**26112**	26096	26154.9	30.4 (-)
X-n359-k29	51505	**51847**	51760	51863.6	39.9 (-)
X-n367-k17	22814	22917	22853	**22911.8**	34.6 (+)
X-n376-k94	147713	148034	147787	**147854.5**	38.4 (+)
X-n384-k52	65928	**66372**	66304	66413.4	47.4 (-)
X-n393-k38	38260	38473	38378	**38449.8**	29.6 (+)
X-n401-k29	66154	66560	66458	**66542.5**	39.7 (+)
X-n411-k19	19712	20104	19862	**20067.7**	64.5 (+)
X-n420-k130	107798	108406	108191	108387.0	66.9 (=)
X-n429-k61	65449	65857	65724	65846.3	59.4 (=)

Table 5.3: Results of 30 runs on the *large* instances (cont.).

	BKS	KGLS	Proposed Algorithm		
Instance	Total Cost	Total Cost	Best	Average	Std. Dev.
X-n439-k37	36391	36534	36434	**36497.5**	31.2 (+)
X-n449-k29	55233	**55753**	55643	55783.3	53.1 (-)
X-n459-k26	24139	24243	24217	24242.3	14.0 (=)
X-n469-k138	221824	223481	223080	**223430.9**	158.5 (+)
X-n480-k70	89449	89972	89833	89965.5	54.9 (=)
X-n491-k59	66483	67189	67062	**67161.8**	48.3 (+)
X-n502-k39	69226	69329	69291	69325.9	20.7 (=)
X-n513-k21	24201	24307	24274	**24305.8**	18.4 (+)
X-n524-k137	154593	157556	156477	157722.0	517.1 (=)
X-n536-k96	94846	**95829**	95714	95867.9	45.7 (-)
X-n548-k50	86700	**86929**	86890	86979.4	44.6 (-)
X-n561-k42	42717	43024	42926	43024.2	57.9 (=)
X-n573-k30	50673	51136	50883	**51006.7**	71.6 (+)
X-n586-k159	190316	191488	191015	**191397.2**	130.6 (+)
X-n599-k92	108451	109327	109136	109335.1	70.4 (=)
X-n613-k62	59535	60235	60098	**60204.7**	61.1 (+)
X-n627-k43	62164	62559	62451	62558.5	43.5 (=)
X-n641-k35	63682	**64074**	64015	64150.3	64.9 (-)
X-n655-k131	106780	**106941**	106936	106969.6	16.5 (-)
X-n670-k126	146332	152154	150463	**151334.0**	550.7 (+)
X-n685-k75	68205	68941	68740	**68890.2**	54.0 (+)
X-n701-k44	81923	**82462**	82356	82486.1	55.0 (-)
X-n716-k35	43373	43875	43713	**43811.6**	46.0 (+)
X-n733-k159	136187	137175	136991	137158.5	94.3 (=)
X-n749-k98	77269	78207	78129	78217.1	48.9 (=)
X-n766-k71	114417	115872	115216	**115765.0**	165.8 (+)
X-n783-k48	72386	73157	73045	**73145.4**	47.5 (+)
X-n801-k40	73305	73642	73485	73654.7	74.2 (=)
X-n819-k171	158121	159711	159396	**159584.0**	78.5 (+)
X-n837-k142	193737	**194943**	194820	195091.7	102.0 (-)
X-n856-k95	88965	**89273**	89217	89318.0	45.9 (-)
X-n876-k59	99299	100152	100072	100146.1	49.9 (=)
X-n895-k37	53860	54377	54230	**54331.9**	50.7 (+)

Table 5.3: Results of 30 runs on the *large* instances (cont.).

	BKS	KGLS	Proposed Algorithm		
Instance	Total Cost	Total Cost	Best	Average	Std. Dev.
X-n916-k207	329179	331241	330643	**331062.8**	153.3 (+)
X-n936-k151	132715	138320	137062	**137845.2**	410.7 (+)
X-n957-k87	85465	**85599**	85707	85786.1	58.1 (-)
X-n979-k58	118976	120035	119962	120062.6	52.7 (=)
X-n1001-k43	72355	73128	72938	73129.5	74.7 (=)

Table 5.4: Results of 30 runs on the *very-large* instances.

		BKS	KGLS	Proposed Algorithm		
Instance	N	Total Cost	Total Cost	Best	Average	Std. Dev.
Antwerp1	6000	477277	483209	481872	**482508.1**	313.7 (+)
Antwerp2	7000	291371	303271	301217	**302041.7**	598.0 (+)
Brussels1	15000	501743	519679	517559	**518447.0**	756.8 (+)
Brussels2	16000	345496	370313	370313	370313.0	0.0 (=)
Flanders1	20000	7240389	7400659	7400659	7400659.0	0.0 (=)
Flanders2	30000	4373440	4641016	4641016	4641016.0	0.0 (=)
Ghent1	10000	469532	478003	475773	**476351.2**	437.3 (+)
Ghent2	11000	257749	274511	271323	**272795.2**	879.6 (+)
Leuven1	3000	192848	194844	194396	**194609.6**	108.7 (+)
Leuven2	4000	111395	116723	115317	**116241.2**	285.4 (+)

From the tables, we can see that the proposed algorithm reaches an improvement or similar quality compared to the KGLS in most medium instances (Table 5.2). Out of the 52 medium instances, 42% of them are better and 25% are equal in quality. The average GAP to the BKS[2] is, however, the same for both methods (both at 0.46%). For the large instances (Table 5.3), there is a slight loss of quality overall, with the average GAP at 0.81% for the KGLS and 0.87% for our heuristic. When looking at the statistical test, however, 68.75% of the instances present a better (+) or similar (=) quality, meaning the difference comes from the minority of instances.

[2]Calculated by: $GAP = (cost - BKS)/BKS$. The smaller GAP the better

As seen in Table 5.4, the very-large-scale shows an improvement for 7 out of the 10 instances. For the remaining 3 instances, there is no significant difference between the proposed algorithm and KGLS, because they both reach the time limit, as 10 minutes is not enough to complete even a full iteration. Additionally, when looking at the column Best, most instances outperform the KGLS. Out of the 110 instances considered, 96 have found a better or same solution than the KGLS.

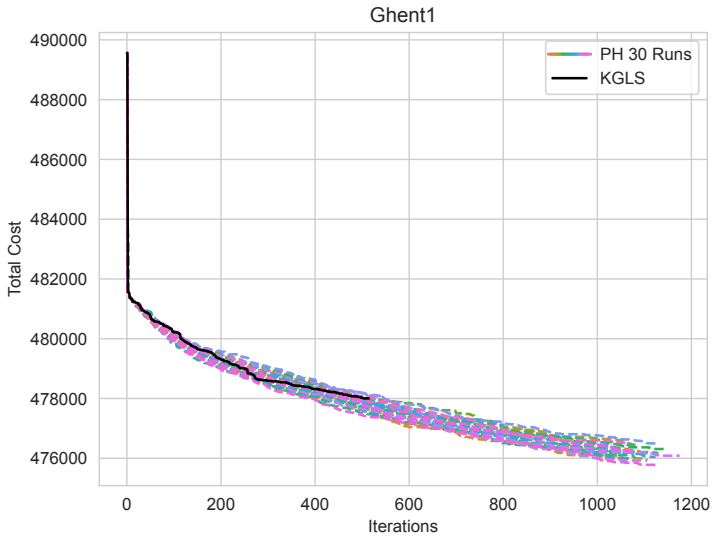


Figure 5.3: Solution progress curve of a single run of the KGLS and the 30 runs of the proposed algorithm (PH) on a *very-large* instance.

A few instances were selected to illustrate some of the observed patterns. In Figures 5.3, 5.4 and 5.5, which show examples of the convergence from the proposed method versus the KGLS in each group (medium, large and very large, respectively). These observations are also seen throughout several instances. The expected behaviour would be similar to the one shown in Figure 5.3, with very similar quality for most of the run-time, but since the pruning allows for a larger number of iterations, it allows for the finding of better solutions. Figure 5.4 finds an interesting outcome of improving the solution much faster than the original, observed by most

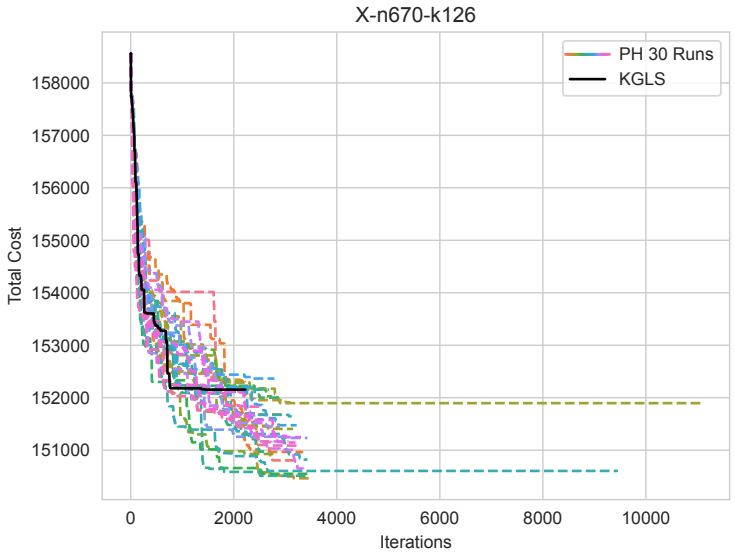


Figure 5.4: Convergence curve of a single run of the KGLS and the 30 runs of the proposed algorithm (PH) on a *large* instance.

runs getting a better result. This could be explained by the fact that the pruning introduces a bit of exploration to the search, by removing possible neighbours that would be selected if considering the same limit. Another interesting fact here are the exceedingly high number of iterations that some runs present. The last case, presented in Figure 5.5, shows what can happen when the pruning is not successful. Even with the extra iterations, it was not able to find a solution better than the KGLS with its original limit for most runs.

5.5 Further Analysis

5.5.1 Efficiency

The improvement on efficiency of the proposed method is quite significant. When comparing the average number of iterations in the same time budget (10 minutes), there is a difference of at least 33% on average, as

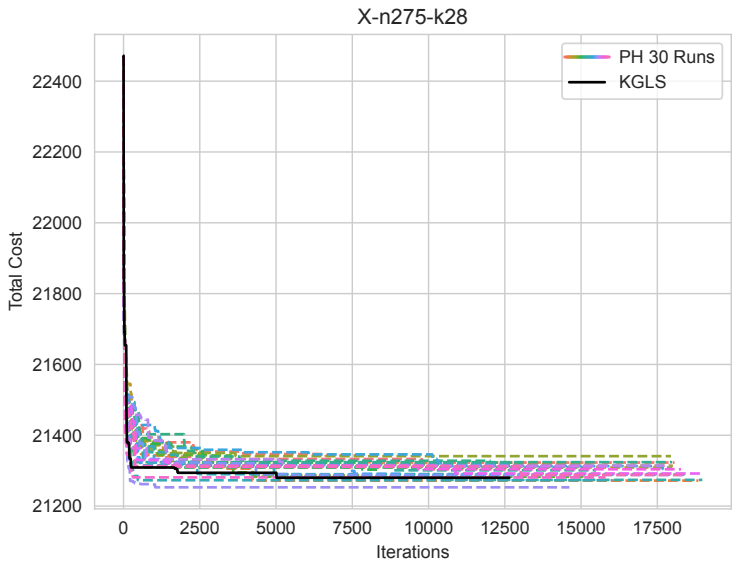


Figure 5.5: Convergence curve of a single run of the KGLS and the 30 runs of the proposed algorithm (PH) on a *medium* instance.

seen in Table 5.5. The difference is even more noticeable for the very-large-scale, being close to 40%. This extra number of iterations allow for a deeper search, which can be beneficial for several search strategies. This is also noticeable in the Figures 5.3, 5.4 and 5.5, where it is clear to see that most runs have more iterations. There are a few outliers with many more, but the majority are within the expected.

Table 5.5: Comparison on the average number of iterations for the KGLS and the proposed heuristic.

Instances	KGLS	Proposed Algorithm	Diff. %
Medium	12914.3	19551.9	33.95
Large	5710.7	9284.1	38.49
Very-large	465.4	773.0	39.79

5.5.2 Performance Instance-wise

In order to determine why some results improved, while others got worse, an analysis by individual instance was performed. This was done by selecting the instances which got worse when using the proposed method, and verifying if there is a pattern regarding the customer distribution or depot location. The customer distribution follow three patterns: Random (R), where the customers are randomly distributed; Clustered (C), where the customers follow cluster patterns; and Random-Clustered (RC), or a mix between both types. As for the depot location, also three possible values: Random (R), Cornered (Co) and Centred (Ce), which are self explanatory. Out of the 100 instances, only 28 had a significantly worse performance with the proposed method[3]. When analysing by the above mentioned categories, it is difficult to point out any standouts. The class "C" with depot in the centre had the most with 50% of the instances getting worse. As the rest of the class "C" performed the best, it could indicate that only this combination is underwhelming. More details are given in Table 5.6. Other than this, there is no clear pattern on why these instances performed worse, at least when looking at these two characteristics.

5.5.3 Neighbourhood Pruning Accuracy

When analysing how far the proposed algorithm is from making the same decision as to the KGLS, the results tell whether or not the decisions are correct most of the time. Table 5.7 summarises the findings. These results can be interpreted as follows: at a given iteration, which best-neighbour that was selected by the KGLS, was also selected by the proposed method? This is considering that at each iteration, only the neighbour with the highest index is saved ($\bar{i}$), for each operator.

[3] As the solutions only got worse for the "X" dataset, to simplify the terminology, only the term instances will be used in this subsection, without differentiating from medium or large.

Table 5.6: Breakdown of instance characteristics for the solutions that got worse.

Customer Distribution	Depot Location R	Ce	Co	Total
R	3/11	1/8	5/16	9/35
C	2/13	5/10	1/9	8/32
RC	4/10	3/14	4/9	11/33
Total	9/34	9/32	10/34	28/100

Table 5.7: Accuracy of the operators CE and RC versus the default pruning size.

Instances	Cross-Exchange (CE)	Relocation Chain (RC)	Total
Medium	97.80%	79.66%	88.73%
Large	96.81%	77.52%	87.16%
Very-Large	90.48%	45.88%	68.18%
Total	96.81%	77.52%	86.86%

Looking at both inter-route operators (CE and RC), since they are tuned independently, there are some interesting patterns to be observed. CE has a higher accuracy rating, meaning that most of the time, the decisions made by the algorithm did not remove the best neighbour out of the search space. When looking at the RC, however, there is a drop in performance. This can be because the RC looks into consecutive routes in the same search, hence a sub-optimal size here would propagate throughout the different routes searched. Despite that, the overall accuracy of the algorithm is quite high, achieving 86.86%. Interestingly, the RC accuracy is even lower in the very-large scale instance, but as shown in Table 5.4, the solution found was better. This is likely since this operators is too expensive, and just reducing its size would allow for more search to

happen, and, consequently, to find better routes. Figures 5.6 and 5.7 exemplify the correctness over 50 iterations (for simplification) by running the proposed heuristic, but also calculating the best neighbour of the original KGLS limit. The dots are the decision made by the KGLS, while the exes are the decision made by the proposed method at the same decision step, the lines were added for visual clarity. As seen, most of the time they are the same, with the few misses that are usually followed from a lower point, which likely guided the limit down.

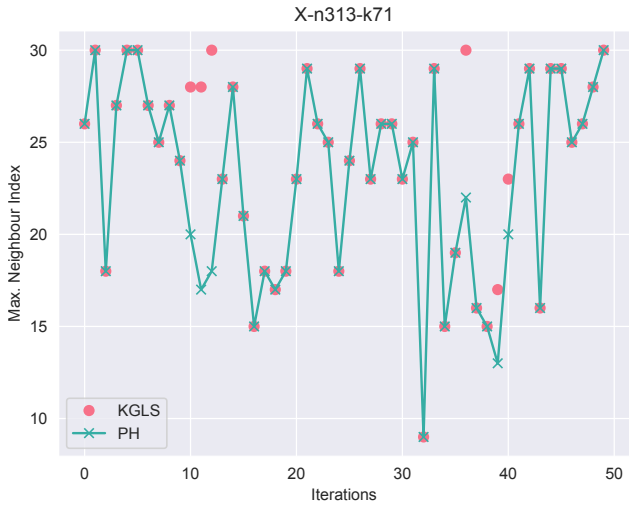


Figure 5.6: Index of the best neighbour for the *CE* operator in each iteration by KGLS and the proposed algorithm (PH).

5.6 Chapter Summary

In this chapter, a novel neighbourhood pruning heuristic is introduced in the context of the LSVRP, combined with the KGLS framework. The idea of the heuristic is to reduce the ceiling of the number of customers to be searched, which would translate to a more efficient search. We achieved this by analysing the (lack of) patterns on the worst-index neighbour (meaning the furthest to the current node) selected for a move in the search

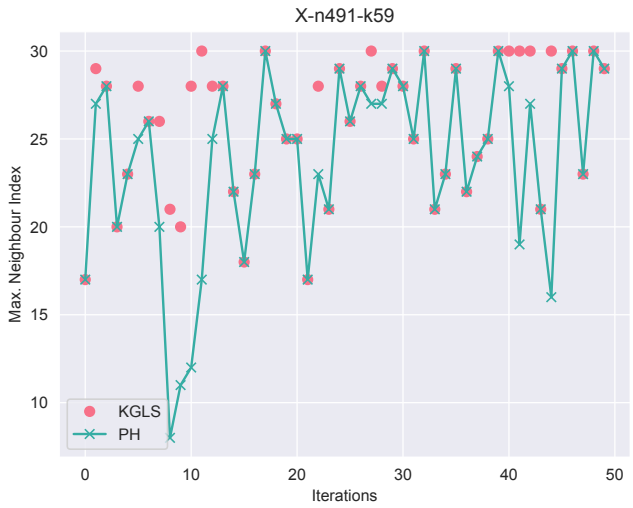


Figure 5.7: Index of the best neighbour for the *RC* operator in each iteration by KGLS and the proposed algorithm (PH).

space. For several instances, it is often that the fixed limit of 30 is not even close to being reached. We add an heuristic which probabilistically reduce this limit to save computational time. The possible drawback is that the limit does not allow better solutions to be found. However, we also add mechanisms to increase the limit again.

Our results show this strategy to very successful, increasing the performance of the KGLS across multiple instances, showing improvements on 87% of the instances in the best case. This extra performance allows the method to find even better solutions within the same amount of time. The results also show that there is still room for improvement regarding automatic pruning of the search space. Some of the presented results and graphs (such as in Figure 5.4) show that by automatically tuning the pruning size, there can be a significant increase of efficiency, which can also translate to an increase in effectiveness.

5.6.1 Consideration on Scalability and Computational time

The approach introduced here fully focused on making the KGLS more scalable, as it reduces the size of the search space. These reductions require minimal overhead (of constant time) and are not affected by the size of the problem. This can be achieved simply by saving the index of the closest neighbour under the search that is already happening. Hence, the computational time is expected to decrease for the same solution quality.

Chapter 6

Learning Strategies to Escape Local Optimum

In this chapter we focus on the acceptance criteria of a local search and how to learn the best one automatically.

6.1 Introduction

In the traditional Local Search framework, the acceptance criteria simply decides whether to accept or not a move. However, the decisions that guide this solution towards this acceptance step can also be considered part of it, especially when we are trying to leave a local optimum. For example, if one search is stuck at a local optimum, the next decision made in order to escape that spot can be attributed as an acceptance step, as the focus of the algorithm changes. Hence, changing its acceptance criteria to allow a worse solution to be accepted. This can lead to modifications in other parts of the framework.

The work of [11] has stepped toward a more rational use of the instance and solution resources as part of the search process. Their deep analysis and investigation of some common characteristics (such as the compactness of a route) for the VRP solutions have led to an efficient method, the

KGLS, in [10]. The KGLS utilises the width of a route as penalisation criteria to guide their search. In total they utilise three distinct penalisation criteria. This penalisation is used when a local optimum is reached. After that, the algorithm enters a penalisation and escape step, where the next selected criteria is used to penalise bad edges. These penalised edges result in a new cost for the solution using a distinct temporary objective function, indirectly forcing a change to the current solution. These changes are reached by applying the same move operators as in the optimisation step, but since the objective function is different, the step will modify the solution.

However, when considering these distinct penalisation criteria across different instances, only a short analysis was made in [10]. Based on few instances, a default penalisation setting was experimentally defined with a specific order of Width, Cost and Width with Cost. Although this penalisation criteria order has shown to be effective, it would be interesting to understand if that is the case for most instances and whether using all of them or not affects the search.

In our experiments, we observed that for some instances using the route width-based penalisation actually damaged the search. Hence, we could explore on when each penalisation criterion is better and gain additional performance on other scenarios. Since this type of problem is more relevant to areas where the cost is important, any savings found are welcomed, especially if it comes from a parameter definition that can be automated to do so efficiently.

In order to tackle this issue and to find how to use the characteristics of the instances being solved to answer it, we will investigate different Machine Learning (ML) techniques, including a novel two-phase multi-tree Genetic Programming (GP) approach. These ML models are first used in a training environment with experimentally generated labels of the best penalisation criterion for each instance, and tested to choose the best KGLS penalisation criterion for unseen instances. We model the penalisation

mode selection criteria as a classification and a regression problem.

6.1.1 Chapter Goals

Considering that these penalisation criteria and its application in the search are an extension of the acceptance criteria, we evaluate this step using the KGLS as our baseline approach. The goals of this chapter are as follows:

1. Understand the relationship between the instance configuration and the best penalisation criterion of the KGLS.

2. Model the prediction as a classification problem (each penalisation criterion has a label), and train classification models on it.

3. Model the prediction as a regression problem (each penalisation criterion has an expected performance metric, i.e., cost improvement over the initial solution by running KGLS with the penalisation) and train regression models on it.

4. Compare the performance of the trained classification and regression models with the baseline KGLS in terms of selecting the penalisation criterion for different instances. Conduct further analysis on the behaviours of the trained models.

6.1.2 Chapter Organisation

The rest of this chapter is divided as follows: Section 6.2 presents the methodology used, while Section 6.3 defines the experimental settings. Finally, results are shown and discussed in Section 6.4 followed by final remarks and future work in Section 6.5.

6.2 Learning Instance-Specific Penalisation Criterion

The idea of developing instance-specific penalisation criterion for the KGLS came from the observation that some instances had a considerable gap between the runs with different criterion. A set of preliminary experiments was performed to verify this on the "X" dataset from [183]. In the experiments, we had set the penalisation criterion to a fixed one and ran it across all 100 instances. These criterion are: KGLS (same as the original work, which alternates, in order, the badness equations 6.1, 6.2 and 6.3), Width-only (considers only the width-based penalisation function 6.1), Cost-only (considers only the cost-based penalisation function 6.2) and Width+Cost (considers the width and cost addition penalisation function 6.3), which will be labelled as K, W, C and WC for simplicity.

$$b^w(i,j) = \frac{w(i,j)}{1+p(i,j)} \tag{6.1}$$

$$b^c(i,j) = \frac{c(i,j)}{1+p(i,j)} \tag{6.2}$$

$$b^{w,c}(i,j) = \frac{w(i,j)+c(i,j)}{1+p(i,j)} \tag{6.3}$$

Although the original mode, class K, has a better success rate overall in average (in a ten minute run the gap to the Best-Known Solution (BKS) is on average 0.52% on the above dataset[1]), this is not true for all instances. To exemplify, we selected a few instances, as shown in Table 6.1, where in bold we have the best value for each instance. The gap between the original criteria (K) and the actual best criteria can be up to 2.5%[2] in the considered experiment. In order to minimise the total cost, if a pre-processing

[1]BKS was taken from the CVRPLib website `http://vrp.atd-lab.inf.puc-rio.br/index.php/en/`

[2]This is calculated by considering if the given criteria is smaller than K. If so, the distance ($F_X - F_K$) is given, where X is the criteria selected.

could find out which criteria was better, then a final better optimisation across all instances could be found. In the perfect scenario, i.e. choosing the best criteria for all 100 instances, the gap to the BKS would fall to 0.43% (from 0.52%), with a total cut on costs of 9.05% (total distances saved over the 100 instances).

Table 6.1: Some examples of the difference in performance between the different penalisation criterion in the KGLS that runs for 10 minutes. Average and Total presented are across all 100 instances from the dataset.

Instance	**K**	**W**	**C**	**WC**	**Largest GAP to K**
X-n153-k22	21953	21953	**21379**	21392	-2.615%
X-n157-k13	16876	16876	16876	16876	0.000%
X-n247-k47	37686	37702	**37676**	37682	-0.027%
X-n308-k13	26166	**26008**	26026	26076	-0.604%
X-n359-k29	51988	51988	**51865**	51894	-0.237%
X-n420-k130	108400	**108315**	108435	108411	-0.078%
X-n449-k29	55772	55821	55803	**55724**	-0.086%
X-n524-k137	157338	**157098**	158185	157978	-0.153%
X-n641-k35	**64086**	64204	64209	64252	0.0%
X-n783-k48	73123	**72960**	73222	73316	-0.223%
X-n979-k58	119832	**119592**	119878	119763	-0.200%
Average	—	—	—	—	-0.126%
Total	—	—	—	—	-9.05%

As can be seen in the data, some of the gaps are very close or even the same, which makes it challenging to learn when each criterion is better. Therefore, it would be easy to create a classifier that would be biased towards a chosen class in case of draw. An alternative is to use a ML method to learn how to predict the improvement given by each criterion. In other words, by changing the problem from a classification to a regression one. Nevertheless, the classification is still considered in our experiments.

Another challenge, and perhaps the biggest one, is determining which characteristics to consider. The features selected will play a big role in

determining whether we can predict the best penalisation criterion for unseen instances. Instance characteristics are the obvious choice, as they are unbiased information about the problem and are promptly available before even searching for an initial solution. However, these characteristics (apart from the very obvious ones like number of customers and total demand) require some expertise on which ones are representative and comparable between different instances. The work of [11] have presented some analysis on instance characteristics, which we are using here. The features are the same instance features used in Chapter 3.3:

- I1 - Number of customers;
- I2 - Number of routes;
- I3 - Ratio between total demand and total capacity available;
- I4 - Average distance between customers' pairs;
- I5 - Standard deviation of the distance between customers' pairs;
- I6 - Average distance from all customers to the central depot;
- I7 - Standard deviation of the distance between all customer to the central depot;
- I8 - Standard deviation of the angle between the customers and the depot (in radians);
- I9* - Initial solution cost provided by CW100[3];

Features I1 and I2 can be extracted directly from the instance (if the number of vehicles is not given, we can simply divide the total demand by a single vehicle's capacity). From the total capacity available from all vehicles, we can calculate feature I3. The next 4 features (I4-I7) are also straightforward, by utilising the distances between the nodes. Feature I8

[3]It is the initial solution algorithm used by the KGLS [9]

considers the angles between the the pair customer and depot, for all customers, which can be calculated with the arc tangent function. Feature I9 is an especial case, because it is only used for the regression process. This feature considers the initial solution cost (considering only the distance).

Although the potential gains appear to be minimal, as shown in Table 6.1 which gives a few examples, we still believe this contribution is necessary in order to further the understanding on VRPs. The potential gains can also be considered as inexpensive to achieve, since it would only require a pre-training process which can be done offline. The outcome would be almost instant, only requiring the calculation of the features for the new instances.

6.2.1 Data Generation

The labels were created by experiments, which show how different criterion perform in given time-frames. The criteria are the different types of penalisation functions used by the KGLS. We run the KGLS considering these different criterion for the same amount of time for the whole dataset. Each penalisation criterion is a possible class label. For example, for an instance A the criteria that found the best value is the Width+Cost mode, then A gets the class label WC, which would be later used for the training process or test accuracy validation. We have run these criterion for all 100 instances of the CVRPLib dataset [183], for 10 minutes and 15 minutes, to verify if the same class persists among runtimes. The results have shown that it continues for most instances (at least 85%) and, therefore, were considered good enough for the rest of the process, considering the 10 minutes the standard for our experiments.

The regression's target variables are the improvement calculated in the initial tests, for each of the classes. This provides a real value which can be used in the regression process. The improvement is calculated based on the initial solution cost, hence why it is fed as one of the input features.

Given an instance and its initial solution constructed by the CW100, the KGLS is ran for a set amount of time. After this time, the difference between the final cost (f) and initial solution cost (i) is calculated such as: $\frac{i-f}{i}$. Since each class produces an independent final cost, we can generate one target for each class, for each instance.

6.2.2 Classification Model

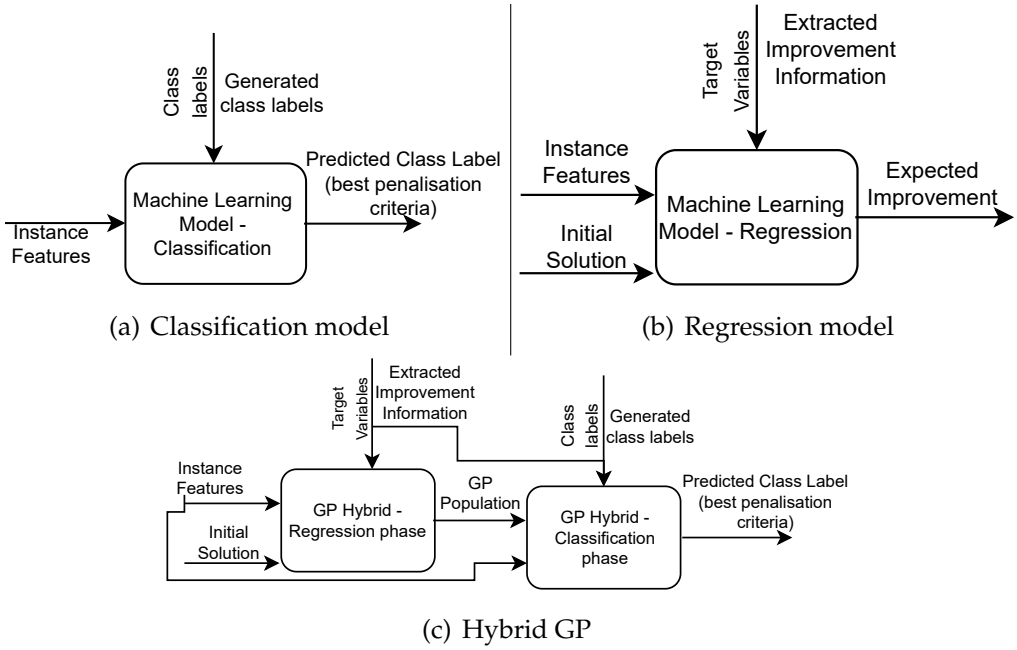
In order to learn the relationship between the features and the class labels by each mode, some commonly used ML algorithms were used: Genetic Programming (**GP**), Support Vector Machines (**SVM**) and Random Forest (**RF**). These methods were chosen due to their efficiency and effectiveness across different scenarios and problems. However, they are sufficiently different for us to make conclusions on a more general level.

The classification model will try to predict which class an instance belongs to. In order to train the ML methods, the features are fed as input and the class labels created are the expected class output. The expectation on these approaches is to learn a correlation between the class and the features. Figure 6.1 (a) summarises the classification input and output.

For the GP approach, a multi-tree individual was used. For this, the individual has 4 distinct trees, one representing each class. For each tree output, the maximum value will indicate which class will be chosen.

6.2.3 Regression Model

The regression model will try to estimate the improvement given by each of the 4 criterion and it considers the same algorithms as the classification approach, since they can easily be adjusted to do regression. For this model, an additional feature is given: the initial solution, calculated by the adapted Clark and Wright savings heuristics, also proposed in the work of [10], based on the classic savings heuristic from [37] (Feature I9, as mentioned above). This feature is exclusive to regression, because by doing so,



(a) Classification model

(b) Regression model

(c) Hybrid GP

Figure 6.1: Training model used for the Machine Learning approaches.

the models have a value that is meaningful to the target variables. They can utilise it to compute the expected improvement in some sort of function. The classification model would not require such since it is only trying to find a correlation. Since the initial solutions can be quickly calculated through the constructive algorithm and, therefore, add a negligible overhead for the whole search process when added (as it will be calculated regardless by the KGLS algorithm).

In the case of the GP, each tree in the multi-tree individual will try to fit one of the classes. The individuals are evaluated by the mean squared error, which should be minimised to approximate the trees' output to the target variables. Hence, the trees that try to learn how to approximate the inputs (features I1-I10) to the improvement. The expectation is that each tree on the best individual after the evolution process, would be able to estimate the improvement given for other unseen instances for each class. Therefore, the greater improvement shown would be the target class for

the execution of the new instance. For example, if tree number 2 (considering the same order mentioned above: K, W, C, WC) has the greatest output from all trees, the class selected would be W. Figure 6.1 (b) illustrates the regression model.

6.2.4 Hybrid Regression-Classification Model for GP

When developing the GP classification approach, it was observed that it would make the decision process more prune to over-fitting, when compared to the regression one. For this, a two-phase model was also considered in order to verify whether we could increase the ability to predict the use of width rather than memorise the input, presented in Figure 6.1 (c).

In general, the algorithm trains the GP model in two phases. The first phase follows the regression model, trying to estimate the improvement given by an initial solution value. The second phase predicts which class is the most suitable for the instance given the estimated value, while also considering the target variables values as part of the fitness function. It does so by calculating the mean squared error, and choosing the tree with the smaller error to be the activated tree (and, therefore, class). Hence in this second-phase the fitness is determined by the classification accuracy.

6.3 Experiments

As already mentioned, the dataset utilised was the CVRPLib [183], since it gives a varied number of instances with different characteristics, such as customers distribution, demand distribution, depot location and number of customers, allowing for both large and ”non-large” variants.

Several experimental settings were tried in order to verify if the methods would learn to classify the instances correctly. Some preliminary experiments (as already mentioned in Section 6.2) were performed in order to verify the difference across all different criterion considered. The KGLS

was run in all criterion for all instances, with two different running times: 10 or 15 minutes. This experiment shows how much each criteria would contribute in the search process if done separately.

After the preliminary experiments, and observing the gap between the criterion, we wanted to check the ability of ML methods to learn this. In order to do so, a parameter tuning was firstly done. This parameter tuning could have several steps, considering the different types of methods and all their possible parameters. However, we decided to keep most of the parameters in the default or most common setting used. Therefore, SVM and RF are utilising the default parameters as implemented in the scikit-library [145], with the exception of the RF, where we limited the max depth to 6. Both of them are also used for both Classification and Regression modes, where they try to match the labels (SVMC and RFC) or learn the improvement (SVMR and RFR).

The GP, however, required a bit of extra tuning, since crossover and mutation rate plays a big role in their performance. We tested two common configurations, as in [8, 200]: 80% crossover rate with 20% mutation rate, and 30% crossover rate with 70% mutation rate. We found the first (80%/20%) was slightly more accurate and were set as the standard.

The dataset required a consideration as well. Although a split in the non-large and large variants can be utilised (there are 21 instances with less than the 200 customer threshold to be considered large-scale, i.e. 79 instances are large-scale), we also considered a larger split to feed more data for the ML algorithms to learn better, dividing the dataset in half (50-50). The parameters are summarised in Table 6.2.

Table 6.2: Parameters considered for the experiments. In bold are the default configuration.

Parameter	**Value**
Population size (GP only)	1000
Initial Population (GP only)	Ramped-half-and-half
Crossover rate (GP only)	80%
Mutation rate (GP only)	20%
Elitism (GP only)	10%
Number of generations (GP only)	100
Number of runs	30

6.4 Results and Discussions

6.4.1 Preliminary analysis on the original data

When analysing the data generated from the KGLS running in all four considered criteria (K, W, C or WC), we looked for any visible patterns on the instance distribution and the better performing class. Although it would be expected for instances with depot on the edge preferably use the criteria where width is considered (based on the somewhat correlation found in [11]), the data does not clearly indicate that. We notice a fairly even split on the instances, as shown in Table 6.3, where the number of instances in which each criteria has performed the best is shown. The values indicate the accumulated difference over those instances against the default criteria K (same as in Table 6.1). Although class K has less instances than class W, K is still better overall. However, this shows that for most instances the best criteria is different from the default one. In fact, if taken all the best criterion, the overall gap from the best-known solution of the algorithm would improve from 0.52% to 0.43%, a 17% improvement.

Table 6.3: Number of instances that each criteria has performed the best. Values represent their relative difference to the default criteria (K). Taken from a 10 minutes execution of the KGLS among all 100 instances.

Mode	Number of instances	Total difference to K
K	28	—
W	29	-3.458%
C	23	-4.148%
WC	20	-1.443%
Total	100	-9.049%

6.4.2 Results for different splits

Another test realised was regarding the amount of training data available for the ML algorithms. Table 6.4 reports these results for both configurations considered: a split between non-large instances and large-scale ones (21:79), and a split of 50% on the instances (50:50). As this is the GAP, the smaller the better. Across all algorithms, the test performance was relatively the same, with the exception of the SVM, which performed slightly better. The results presented are the GAP to the BKS, but since the target best is different, we also present for each mode. Although namely the first split (21:79) looks better, we have to compare to this target. The difference (how far from the target each method is) between the methods for both splits are less than 1% on average.

This, perhaps, is the most intriguing result. Although it was initially intended to discover if the bad performance in the tests (which only considered 21:79 split) was because of a lack of data, this test shows that the problem is within the data. As giving more data actually made it slightly worse. Hence, we chose to keep using the 21:79. These results indicate that the features cannot comprehend the difference on the data.

Table 6.4: Summary of the performance when comparing two different splits on the dataset. Values are **GAP** to the BKS, averaged for the 30 runs and the respective instances. Column Best is the best run out of the 30.

ML Method	Data Splits			
	21:79		50:50	
	Average	Best	Average	Best
Best Penalisation		0.503%		0.540%
GP Classification	0.622%	0.588%	0.672%	0.628%
GP Regression	0.616%	0.572%	0.663%	0.625%
GP Hybrid	0.621%	0.572%	0.664%	0.608%
SVM Classification	0.626%	0.626%	0.617%	0.617%
SVM Regression	0.609%	0.609%	0.696%	0.696%
RF Classification	0.614%	0.602%	0.656%	0.631%
RF Regression	0.626%	0.608%	0.663%	0.642%

6.4.3 Training Performance

When looking at the training performance, we observed a fairly good accuracy: 84% for the GP classification, an error of 0.006 for GP Regression and 75% for the hybrid mode, on average for the 30 runs. Random Forest in classification mode performed the best, with accuracy of almost 100%, and the regression one was not much back. While SVM did not seem to be able to separate the instances very well.

However, as pure accuracy and error are not the end goals of our task, we need to compare the methods regarding our optimisation goal. Table 6.5 summarises the training models' performance, in this aspect, where the actual distance to BKS is compared. This show whether the methods can pick the correct expected criteria during training, or even if not, how much worse would they turn out to be. When analysing by these measures, the results show the RF as a clear winner, while the GP methods were able to find best individuals with very similar performance. Both RF and GP can get better training performance than the baseline methods. We expect this would indicate that the methods are able to learn how to correlate the

input data to the output. However, as the next experiment shows, it is not that straightforward.

Table 6.5: The training results. Presented values are the distance to the BKS, averaged out across the training data. Columns 1-4 shows the values for the KGLS criterion. Column Best shows the target value for the ML methods. Next columns represent the average and best over the 30 runs for the ML algorithms studied.

						SVM		RF		GP		
K	**W**	**C**	**WC**	**Best**		**Class.**	**Reg.**	**Class.**	**Reg.**	**Class.**	**Reg.**	**Hybrid**
					Average	0.39%	0.38%	0.16%	0.17%	0.38%	0.29%	0.30%
0.33%	0.38%	0.27%	0.28%	0.16%	**Best**	0.39%	0.38%	0.16%	0.16%	0.25%	0.18%	0.20%

6.4.4 Test Performance

As Table 6.6 shows, the results across all the ML models test performance are worse than that of the default criteria (K), with the best case of both the GP-Regression and GP-Hybrid present a similar quality as K. This conflicts with the training data results, which would show the training was able to learn the data (except for SVM, which always chooses either one or two classes). This seems like a clear indication of over-fitting. However, as the previous experiments showed, increasing the amount of data did not improve the models performance significantly.

To analyse why the problem did not improve when increasing the number of instances and why the method does not appear to maintain performance from the training to the test, we had to look at the data more closely. Figure 6.2 is an example of a plotted instance improvement values and one run attempt to match it for a random single GP run. The red cross shows that the regression variable targets are too close to each other (improvement is calculated by the relative difference between final and initial solution, as explained in Section 6.2.1). On the other hand, the methods are not able to learn the patterns with precision for most cases, as seen by the

Table 6.6: The results for a subset of the test set. Presented values are the distance to the BKS. Columns 2-5 shows the criteria of the KGLS run. Best shows the best values from each criterion. Next columns represent the average expected GAP over the 30 runs for the ML algorithms. Finally, last rows present the average and best results over the 79 test instances.

						SVM		RF		GP		
Instance	**K**	**W**	**C**	**WC**	**Best**	**Class.**	**Reg.**	**Class.**	**Reg.**	**Class.**	**Reg.**	**Hybrid**
X-n209-k16	0.25%	0.18%	0.19%	0.23%	0.18%	0.19%	**0.18%**	0.19 %	0.25 %	0.19 %	0.21%	0.22%
X-n247-k47	1.11%	1.15%	1.08%	1.09%	1.08%	1.15%	1.15%	**1.09%**	**1.09%**	1.12%	1.11%	1.10%
X-n266-k58	0.64%	0.49%	0.60%	0.67%	0.49%	**0.49%**	**0.49%**	0.50%	0.64%	0.55%	0.59%	0.63%
X-n280-k17	0.56%	0.58%	0.45%	0.39%	0.39%	**0.45%**	0.58%	0.50%	0.56%	0.55%	0.50%	**0.45%**
X-n308-k13	1.19%	0.58%	0.65%	0.84%	0.58%	0.65%	**0.58%**	0.64%	1.19%	0.63%	0.77%	0.83%
X-n317-k53	0.07%	0.09%	0.06%	0.08%	0.06%	**0.06%**	0.09%	0.07%	0.07%	0.08%	0.08%	0.08%
X-n359-k29	0.93%	0.69%	0.83%	0.75%	0.69%	0.83%	**0.69%**	0.93%	0.93%	0.70%	0.77%	0.78%
X-n367-k17	0.45%	0.34%	0.28%	0.32%	0.28%	0.34%	0.34%	**0.28%**	0.30%	0.34%	0.34%	0.33%
X-n420-k130	0.56%	0.48%	0.59%	0.57%	0.48%	**0.48%**	**0.48%**	**0.48%**	0.57%	0.52%	0.54%	0.56%
X-n449-k29	0.75%	0.84%	0.80%	0.66%	0.66%	0.80%	0.84%	0.75%	0.75%	0.77%	0.78%	**0.71%**
X-n469-k138	0.66%	0.63%	0.75%	0.71%	0.63%	0.75%	**0.63%**	0.68%	0.66%	0.67%	0.69%	0.70%
X-n524-k137	1.77%	1.62%	2.32%	2.19%	1.62%	**1.62%**	**1.62%**	**1.62%**	1.74%	2.02%	1.97%	2.05%
X-n573-k30	0.19%	0.41%	0.25%	0.25%	0.19%	0.25%	0.41%	0.25%	**0.24%**	0.36%	0.29%	0.26%
X-n627-k43	0.31%	0.22%	0.34%	0.28%	0.22%	0.34%	**0.22%**	0.25%	0.31%	0.26%	0.27%	0.29%
X-n685-k75	0.66%	0.78%	0.58%	0.60%	0.58%	0.78%	0.78%	0.78%	**0.60%**	0.69%	0.68%	0.64%
X-n716-k35	0.68%	0.55%	0.87%	0.66%	0.55%	0.87%	**0.55%**	0.87%	0.71%	0.60%	0.66%	0.69%
X-n783-k48%	0.54%	0.32%	0.68%	0.81%	0.32%	0.68%	**0.32%**	0.56%	0.54%	0.57%	0.55%	0.70%
X-n819-k171	0.59%	0.82%	0.71%	0.61%	0.59%	0.82%	0.82%	0.82%	**0.61%**	0.74%	0.69%	0.66%
X-n856-k95	0.31%	0.26%	0.26%	0.34%	0.26%	**0.26%**	**0.26%**	**0.26%**	0.34%	0.29%	0.28%	0.30%
X-n895-k37%	0.20%	0.32%	0.43%	0.34%	0.20%	0.43%	0.32%	0.32%	**0.22%**	0.33%	0.33%	0.34%
X-n936-k151	2.98%	3.54%	2.95%	3.89%	2.95%	3.54%	3.54%	3.54%	3.79%	3.44%	**3.33%**	3.52%
X-n979-k58	0.54%	0.33%	0.57%	0.48%	0.33%	0.57%	**0.33%**	0.57%	0.54%	0.41%	0.45%	0.48%
X-n1001-k43	0.40%	0.27%	0.52%	0.50%	0.27%	0.52%	**0.27%**	0.28%	0.40%	0.39%	0.38%	0.45%
Average	**0.572%**	0.609%	0.644%	0.632%	0.503%	0.626%	**0.609%**	0.614%	0.626%	0.622%	0.616%	0.621%
Best	—	—	—	—	—	0.626%	0.609%	0.599%	0.607%	0.588%	0.572%	0.572%

example in Figure 6.2, represented by the blue dots. The blue dots seem to have a fairly different pattern than the target values, while also being a bit off regarding the actual value. This behaviour is also observed across several instances. Hence, a small mistake there would make the accuracy drop by selecting a different class, which could affect the training process. Nonetheless, the closeness of the targets variables would likely make it difficult for the ML methods to learn the difference between the instances.

When analysing the case-by-case performance instead of the average, as the average might be skewed towards the worse solutions (i.e. solutions

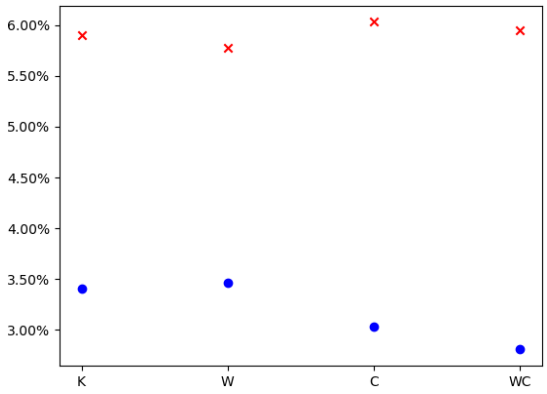


Figure 6.2: An example of training instance considered. The red exes represent the actual improvement values, while the blue dots represent the GP regression prediction.

which contribute the most to the average GAP), we start to see a better performance scenario. As Table 6.7 shows, out of the 79 test instances, most methods seem to choose about half correctly. The best case for each shows that both the RF and GP regression models have selected a majority correctly. As the best-case average GAP is not near the target value in those cases, this can indicate that the wrongly chosen ones are damaging the average significantly.

Table 6.7: Number of instances the models have outperformed the default criteria K, out of the 79 test instances. Average and best case over 30 runs.

	SVM		**RF**		**GP**		
	Class.	**Reg.**	**Class.**	**Reg.**	**Class.**	**Reg.**	**Hybrid**
Average	27	33	32	41	28	28	30
Best	27	33	40	45	35	59	34

Finally, to confirm that the results were significantly different among the ML methods, we applied the Friedman statistical test and verified that the p-value was ≤ 0.001. This indicated there is difference in the performance. Then, we ran the Wilcoxon Rank-Sum test pairwise, which showed

the difference were mostly related to the SVM performance. Test performance across the ML methods is shown in 6.3.

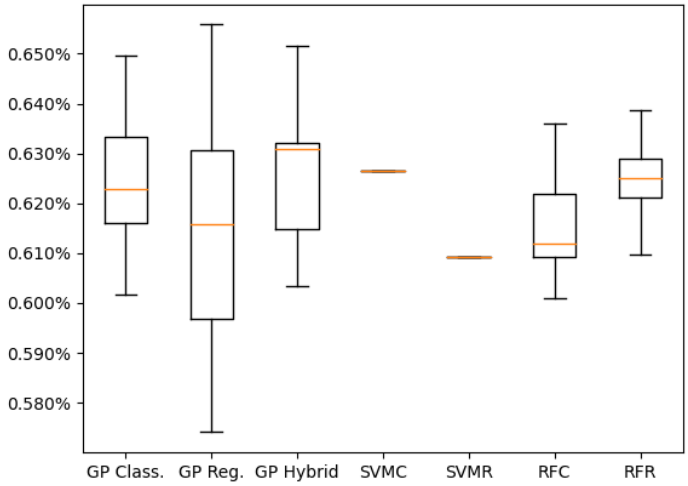


Figure 6.3: Box plot for the performance of the different ML method used with outliers removed. The values indicate the distance to the BKS, therefore, the lower the better.

6.5 Chapter Summary

In this chapter we introduce a Machine Learning approach to learn and predict which penalisation criteria would be more successful in the KGLS framework. This application as part of the acceptance criteria helps guide the solution out of local optima. We first introduce how the original method performs across different instances with the distinct criterion being applied. We show that there can be a considerable gap between the criteria.

We then apply three distinct learning processes: classification, regression and a hybrid approach. Each approach utilises the instance features in order to predict which criteria will perform better for the specific solution. We label the data experimentally and apply the methods on unseen data. Showing that we are able to predict the best method in several occasions,

however not consistently enough. The main drawbacks observed come from the lack of distinction in the labelled data (for both classification and regression), since for most instances the solutions are very similar, making it very hard for the ML algorithms to learn.

6.5.1 Consideration on Scalability and Computational time

Similarly, as the first approach in Chapter 3, the approach introduced here is safe regarding the scale of the instance being solved. The calculation of features has very little difference for larger sizes. The trained models will have the same input size regardless of instance size. The computational time will also be similar to original KGLS as fixing the penalisation strategy does not affect the efficiency of the algorithm.

Chapter 7

Conclusions

The overall goal of this thesis is to improve on *the automation of different steps of local search-based metaheuristics for the Large Scale Vehicle Routing Problem*. This target was successfully achieved by breaking down the main tasks of a local search framework and automating them individually, each contributing to the whole. These tasks include the initialisation stage, through the use of ML and GPHH techniques, the improvement stage, through hyper-heuristics and online learning, and, acceptance and evaluation stage, through classification and regression approaches. The employed techniques have shown improvement in effectiveness or efficiency, and sometimes both, in solving the LSVRP. And even though some of the results are underwhelming, we managed to improve HHs enough to make them competitive with other state-of-the-art metaheuristics. The computational time added to the search of the best heuristic, however, can be a bit high in several of the HH frameworks introduced (hours to a day of training, depending on the scale of the problem), especially the ones introduced in Chapter 4. But considering some applications can spend such training time, they can still be useful in some scenarios, for example, finding a set of routes that will be executed several times.

The rest of this chapter is divided as follows: we discuss the objectives from each contribution chapter and their overall achievements. Then, we

expand on a high level discussion on the conclusions of each chapter. Finally, we discuss possible future work and research directions in the scope discussed here in this thesis.

7.1 Research Questions and Main Conclusions

In this section we bring back the questions asked in our Introduction chapter (Chapter 1.3) and evaluate whether we were able to successfully answer them. We present the following objectives, respectively for each contribution chapter (Chapters 3 - 6):

7.1.1 Learning Effective Initialisation

Our first objective in this thesis was to **develop automatic initialisation methods improving the effectiveness and efficiency of a local search-based metaheuristics for the LSVRP**. This objective was presented in Chapter 3, we achieved it by considering two different approaches.

In Chapter 3, we argue that the initialisation heuristic plays a role in the performance of Local Search-based methods and should not be neglected when developing solution approaches. We present evidence that even with very bad costs some initial solutions can still find better final solutions than the ones with a good initial cost. For example, we observe that the knowingly bad Nearest Neighbour heuristic ends up having the best performance overall by the end of the KGLS run. Although we do not claim this observation is generalisable for other metaheuristics, we believe similar behaviour could be found for other frameworks if the same methodology is applied. When observing the difference in solution quality among different initial heuristics, we also utilise features from the solutions and instances to train 10 ML methods to learn how to identify the cases in which each of the four considered constructive heuristics can perform better. The different ML techniques show that there is some correla-

tion between the features selected and the ability to predict which method would find better performance as if learning how to predict in which part of the search space the selected KGLS would perform better. Hence, we can claim that learning to predict the initialisation heuristic improves the *effectiveness* of our local search-based metaheuristic.

In Chapter 3, we also propose a Genetic Programming Hyper-Heuristic to construct LSVRP solutions in a specific region of the search space. The new proposed fitness function guides the solutions towards routes with less width, which have shown to be a good indicator of near-optimal solutions. The method is able to produce a solution that can be improved towards better final solutions compared to a traditional heuristic method in 49 out of the 79 test instances. Additionally, the proposed GPHH was able to achieve the best solution in the presented tests for a few instances, even when using simpler heuristics than the state-of-the-art. Therefore, we can also show an increase in *effectiveness* in some scenarios. The *efficiency* can be argued when observing some of the better solutions are found faster.

The questions initially asked in our research were:

***1.* How much impact does an initial solution of a local search-based metaheuristic have? *2.* And can we learn to utilise this solution in favour of our search?**

We answer the first question by providing an initial solution comparison, proving that there is a difference in performance that can be significant, simply by changing the initial solution — and that this difference cannot be correlated to only the ***cost*** of the solution. We utilise that knowledge to learn how to select the initial heuristics and that, although relatively small, they provide a reduction in costs across the majority of the instances tested, which answers the second question.

The second question can be answered by considering our GPHH approach, which shows it is possible to build a solution in a more promising region of the search space. Although this approach has weaker results, it was still able to find some better solutions despite having both a weaker

framework and a worst starting solution.

7.1.2 Operator Selection and Bounds with Evolutionary Hyper-Heuristics

The second objective of this thesis was **to develop Selection HHs using a genetic algorithm with new chromosomes which incorporate a local search-based method improvement step, including efficient search space limits for the LSVRP**. This objective was also achieved and we present it in Chapter 4. For this goal, three different strategies were developed, each with its own particularities.

The first strategy proposes an adaptive clustering technique based on solution evolution in combination with a Selection HH GA-based framework. The experiments realised showed the effectiveness of both the clustering technique introduced, as well as its combination with a GA for selecting low-level heuristics. The results presented have shown that the clustering technique significantly improves the performance when compared to scenarios with no search limits or with fixed clusters, which would be the more traditional approach. Therefore, the technique finds its purpose of *limiting the search space more efficiently*, as in our goal.

Chapter 4 also presented a GA HH with a novel two-level chromosome that limits the search space for each operator in an attempt to find more effective neighbourhood sizes. The experiments realized have shown the method's ability to find better solution quality than the compared manually designed approaches for all the test cases. And when compared to a fixed limit Hyper-Heuristic, it also improved for most cases. We also successfully show that not only the order of which operator are applied impacts the solution, but also the sizes for each neighbourhood. Hence, this new chromosome was able to *improve the effectiveness and efficiency* in this improvement step.

A third approach is also introduced in Chapter 4, where a Selection

HH is used to automatically configure the KGLS algorithm, as well as its heuristics' pruning, introducing another novel chromosome which incorporates all three main stages of the local search. The results, however, are underwhelming as they fail to find competitive results when compared to the original KGLS. This can be explained by the big search space considered, since having more layers and possible configurations, the heuristic search space became too expensive to explore considering the scenarios presented. Although this happens, we are confident that the goal of providing a HH method that can automatically and effectively prune the solution search space was achieved in this chapter, as all three methods provide an insight or result that confirms their effectiveness or efficiency.

Considering the research questions asked for this objective:

***1.* Does the order of the neighbourhood operators affect the search? If so, *2.* can we optimise it? And *3.* can we automatically incorporate proper limits into their neighbourhood scope without reducing their effectiveness?**

We answer the first question by showing that the order of the operators affected the performance when comparing to a fixed order. However, this order is not something generalisable, as it depends on both the starting point and the type of operators used. We are able to show that, and answering question 2, some orders are more beneficial for some instances and could be optimised individually.

The third question is answered by the different pruning strategies being successfully employed. The use of adaptive clusters and operator-dependent pruning limits were shown to be the most successful part of our approach and all strategies were able to find more efficient exploration.

7.1.3 Learning to Guide Solution Search Space

In the third objective of this thesis we wanted to introduce a **new strategy that adaptively changes the search limits for each inter-route operator**

when solving the LSVRP. Chapter 5 achieves this objective by presenting a novel neighbourhood pruning heuristic.

The heuristic is used to reduce the ceiling of the number of customers to be searched, translating to a more efficient search. The proposed method was able to achieve better performance than the baseline KGLS with fixed neighbourhood size on most of the instances, especially the very large ones. Additional analyses have showed that the heuristic was able to capture the best neighbour in most cases, about 87% in total. This high accuracy together with the *improved efficiency* indicates that the proposed method was able to adapt the neighbourhood size as intended by our objective.

Another note on the approach is that it did not add expensive overhead to learn this decision-making process. This can be seen by the results, which show that not only it did not damage the search, but was also able to increase its the efficiency and effectiveness.

Bringing back the research question that led us to this objective:

***1.* Can we limit the neighbourhood search space for each operator adaptively? *2.* Does that improve the efficiency? *3.* Does it come with a loss of effectiveness?**

Question 1 can be safely answered, as each operator's limit is individually tailored by the proposed heuristic strategy. The results corroborate question 2, showing an increase in the number of iterations quite significantly in the same time-frame.

Question 3 is also shown to be true for most cases, as the method introduces a way to reduce the limit and still outperformed the original limit. This implies the limits used most of the time are quite large and could be reduced without loss of effectiveness. However, that is not true for all cases, as some instances had a decrease in performance, meaning they operate on the limit of the search pruning. Hence an adaptive strategy seems to be more reliable, as it is less likely to get it wrong for specific scenarios. Another argument that can be seen as a consequence of the pruning em-

ployed, is that changing the pruning limit may just be leading the search direction to a different best neighbour at each stage. However, this also might benefit the search as exploration, also allowing it to reach a better final solution than the KGLS — thus effectively keeping its ability to find competitive solutions.

7.1.4 Learning Acceptance Strategies to Escape from Local Optima

In our fourth and final objective we aimed to **introduce the use of ML techniques to differentiate the most-suitable acceptance criteria and to build novel ones, increasing the effectiveness in the search for better LSVRP solutions**. This objective was partially achieved in Chapter 6, being able to predict the best penalisation criteria in a KGLS framework.

In Chapter 6 we attempt to find the best improvement in the KGLS based on observations of differences in quality between distinct penalisation criteria. Although the difference can be small, it can still be translated to a cut in costs, especially for larger scenarios. The presented analysis tries to show that there is a strong disconnection between what we perceive (instance characteristics), to what is actually useful. Although there seem to be correlation on some instance characteristics and the use of width, as pointed in the work of [11], the utilised features are not enough to make such correlation generalisable.

We considered different ML techniques, configurations and labelling methods, and we still could not confidently and correctly predict the criteria in which an instance should be run when executing the KGLS algorithm. However, in the best case, we were able to choose more than half correct instances, indicating that a model should exist which can make a more accurate prediction. It also seems that the instances characteristics do not provide enough information for the corresponding classes.

When we go back to the research questions related to this objective:

***1.* Can we learn to automatically select the acceptance criteria? *2.* Is having a pool of criteria beneficial? *3.* Can we automatically learn a new method to drive out of local optima?**

Our first question is partially answered by our ML approach, as we had some ability to predict the penalisation criteria — which we argue as one of the main stages of the acceptance criteria in the case of the KGLS. Although the somewhat poor results in the prediction, question 2 is answered by showing the difference in performance for different criteria, where each criterion is able to outperform the others for some instances, implying a pool can be beneficial. Question 3 was not fully answered as we were not able to extend the method's ability to generalise the learning in the first place, not allowing for knowledge to be extracted in order to build new methods to provide a more effective local optima escape.

7.2 Future Work

In this section we present what future directions our research can go in. We present them according to the contribution chapters of this thesis, followed by new directions that we judged important to address, but are not necessarily connected to the contributions given.

7.2.1 Learning Effective Initialisation

The results found in Chapter 3 encourage us to explore more on the effects in which the initial search space has on the performance of the local search heuristics. Although we use only one specific local search-based method, we believe that the techniques presented are transferable to other local search-based methods with minor adjustments. A relevant next step that can be used across the different methods introduced, would be to explore in more detail how each feature influences the final classification or performance of the considered methods. A fine-tuning on the parameters

and a possible reduction in the feature space size or the addition of more relevant ones are also good next steps across both methods.

Another interesting use for the ML technique is to rank solutions from a population-based metaheuristic, increasing their selection chance as they could be likely to provide better offspring. For example, combining the recent GA from [184] with our ML approach, as the added overhead being small with a limited number of features that needs to be recalculated at each generation, it would be possible to use a trained model in order to select more promising parents, rather than using the traditional tournament selection which only takes into account the total cost.

When considering a immediate next step for this approach, however, the use of more construction methods would be the obvious choice. This addition would have to consider if the gain in search performance outgains the loss of time in calculating all initial solutions. The use of our GPHH approach, also introduced in Chapter 3, did not seem appropriate as its construction time is relatively slow when compared to the other constructive heuristics — and as it also lacks robustness.

For the GPHH approach, one clear weakness comes from the way the routes are built, requiring a more competitive and efficient route construction method to be used. As GP has a large search space and we opted for aiming to build generalisable trees, we might have lost some of its potential by including several instances in a single training evaluation. A future work would do a better job in training the GPHH, especially since training time can be disregarded, as long as it achieves a good result in the end.

7.2.2 Operator Selection and Bounds with Evolutionary Hyper-Heuristics

Chapter 4 explores the automatic configuration of the improvement step via evolutionary hyper-heuristics. The common challenge in the three methods considered is the evaluation of each chromosome. It is an ex-

pensive task to run a local search just to evaluate it, especially considering the limited time given might not be enough to calculate each individual's true fitness. To answer this challenge, surrogate techniques which can estimate the fitness cost in a fraction of the time could be a decisive point to make such approach more usable. Another consideration would be to investigate whether evolving the population for multiple instances could lead to a chromosome applicable to more cases, rather than training for a single instance. The genetic components can also be tuned to increase the efficacy of the evolutionary processes.

In our cluster-based approach, the parameter M is a weak-point, as it requires tuning. We believe a similar approach to what was done in Chapter 5 could be done, adaptively changing the parameter according to recent progress. The lack of local optima escape mechanisms affects the measurement of potential for this and the second approach.

For our second and third approaches, we notice that fine tuning the percentage of limits is a difficult task. An idea to improve that is to apply a local search to optimise the limits separately or through the use of a separate phase for tuning them — as it showed to have a poor performance in doing both the LLH and their limits simultaneously.

7.2.3 Learning to Guide Solution Search Space

In Chapter 5 we explore an adaptive approach for automatically setting up the limit for each neighbourhood operator. Future work in this research direction could be on to how to improve the accuracy of this type of pruning. Removing the expertise element in the heuristic's decisions is perhaps the most obvious one, allowing the method to be fully automatic. This can be done either by using statistical learning or by using some non-supervised learning technique. Another consideration is to utilise this method with a different framework since the proposed heuristic is generic and mostly independent of the underlying algorithm. For example, rather than the

fixed approach utilised in [184], which limits the search for the 20 closest customers, we could apply the same method to improve its efficacy.

7.2.4 Learning Acceptance Strategies to Escape from Local Optima

Finally, in Chapter 6 we utilise ML to predict the best penalisation criteria for the KGLS local-optima-escape phase. In future work, introducing more features might help to distinguish the classes better, especially if they are related to how the penalisation works. Increasing the number of instances to train and test on, utilising an automatic method to generate instances, would provide more details to the ML methods on how to separate the instances more effectively. Another idea would be to separate the instances which present very similar results from the ones that do show some gaps between different criteria. Doing so would be adding a bias observed on the seen data and, hence, was not done in this work. However, this type of strategy could help us identify the type of features that can be further explored by other approaches.

7.2.5 New Directions

Among the future work that do not relate to our contributions directly, we can say the use of techniques such as GP to build the operators is one of them. Although the automatic selection of low-level heuristics through hyper-heuristics or similar strategies minimise the manual design decisions, the operators themselves are still manually designed. Generating new ones have been attempted with different techniques, but seldom producing any worthy results and, as far as the authors are aware, not built considering large-scale. Using GP (or even Linear GP, a variant of GP that allows graphs rather than trees) we could generate heuristics that can be used for specific instance configurations or customer types, while also considering the efficiency of the built operator, not only its effectiveness.

Another possible new direction comes from the use of GP to generate new penalisation modes for GLS-based methods. For example, considering the extensive use of domain knowledge to generate the penalisation functions used by [10] (as presented in Chapter 2.5.4), we could use the ability to generate new functions of GP to do so automatically. This would allow for new combinations of cost and width, or even something unrelated, that can be more effective in the escape of local optima, improving the quality of any GLS-based method in the given context.

Chapter 8

Bibliography

[1] AARTS, E., AND LENSTRA, J. K. *Local Search in Combinatorial Optimization*. Princeton University Press, 2003.

[2] ACCORSI, L., LODI, A., AND VIGO, D. Guidelines for the computational testing of machine learning approaches to vehicle routing problems. 229–234.

[3] AGÁRDI, A. Fitness landscape analysis of population-based heuristics in solving a complex vehicle routing problem. In *Vehicle and Automotive Engineering 4* (2023), K. Jármai and Á. Cservenák, Eds., Lecture Notes in Mechanical Engineering, Springer International Publishing, pp. 667–677.

[4] AGARWAL, Y., MATHUR, K., AND SALKIN, H. M. A set-partitioning-based exact algorithm for the vehicle routing problem. *Networks 19*, 7 (dec 1989), 731–749.

[5] AL-SAHAF, H., BI, Y., CHEN, Q., LENSEN, A., MEI, Y., SUN, Y., TRAN, B., XUE, B., AND ZHANG, M. A survey on evolutionary machine learning. *J. Roy. Soc. New Zeal. 49*, 2 (2019), 205–228.

[6] ALESIANI, F., ERMIS, G., AND GKIOTSALITIS, K. Constrained clustering for the capacitated vehicle routing problem (cc-cvrp). *Applied Artificial Intelligence* (Jan. 2022), 1–25.

[7] ARCHETTI, C., AND SPERANZA, M. G. The split delivery vehicle routing problem: A survey. *The Vehicle Routing Problem: Latest Advances and New Challenges* (2008), 103–122.

[8] ARDEH, M. A., MEI, Y., AND ZHANG, M. Transfer learning in genetic programming hyper-heuristic for solving uncertain capacitated arc routing problem. In *CEC* (2019), IEEE, pp. 49–56.

[9] ARNOLD, F., GENDREAU, M., AND SÖRENSEN, K. Efficiently solving very large-scale routing problems. *Computers & OR 107* (2019), 32–42.

[10] ARNOLD, F., AND SÖRENSEN, K. Knowledge-guided local search for the vehicle routing problem. *Computers & Operations Research 105* (may 2019), 32–46.

[11] ARNOLD, F., AND SÖRENSEN, K. What makes a vrp solution good? the generation of problem-specific knowledge for heuristics. *Computers & Operations Research 106* (2019), 280–288.

[12] BAI, R., CHEN, X., CHEN, Z.-L., CUI, T., GONG, S., HE, W., JIANG, X., JIN, H., JIN, J., KENDALL, G., LI, J., LU, Z., REN, J., WENG, P., XUE, N., AND ZHANG, H. Analytics and machine learning in vehicle routing research. *International Journal of Production Research 61*, 1 (2023), 4–30.

[13] BALDACCI, R., HADJICONSTANTINOU, E., AND MINGOZZI, A. An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research 52*, 5 (oct 2004), 723–738.

[14] BALINSKI, M. L., AND QUANDT, R. E. On an integer program for a delivery problem. *Operations Research 12*, 2 (apr 1964), 300–304.

[15] BANG-JENSEN, J., GUTIN, G., AND YEO, A. When the greedy algorithm fails. *Discrete Optimization 1*, 2 (Nov. 2004), 121–127.

[16] BAXTER, J. Local optima avoidance in depot location. *Journal of the Operational Research Society 32*, 9 (sep 1981), 815–819.

[17] BELL, J. E., AND MCMULLEN, P. R. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics 18*, 1 (jan 2004), 41–48.

[18] BELLMORE, M., AND NEMHAUSER, G. L. The traveling salesman problem: A survey. *Operations Research 16*, 3 (1968), 538–558.

[19] BENGIO, Y., LODI, A., AND PROUVOST, A. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research 290*, 2 (Apr. 2021), 405–421.

[20] BOCHTIS, D., AND SØRENSEN, C. The vehicle routing problem in field logistics part i. *Biosystems Engineering 104*, 4 (Dec. 2009), 447–457.

[21] BOCHTIS, D., AND SØRENSEN, C. The vehicle routing problem in field logistics: Part ii. *Biosystems Engineering 105*, 2 (Feb. 2010), 180–188.

[22] BRAEKERS, K., RAMAEKERS, K., AND NIEUWENHUYSE, I. V. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering 99* (sep 2016), 300–313.

[23] BRANKE, J., NGUYEN, S., PICKARDT, C. W., AND ZHANG, M. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation 20*, 1 (Feb. 2016), 110–124.

[24] BRÄYSY, O. A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS Journal on Computing 15*, 4 (nov 2003), 347–368.

[25] BRÄYSY, O., AND GENDREAU, M. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science 39*, 1 (feb 2005), 104–118.

[26] BREITBARTH, E., GROβ, W., AND ZIENAU, A. Protecting vulnerable people during pandemics through home delivery of essential supplies: a distribution logistics model. *Journal of Humanitarian Logistics and Supply Chain Management 11*, 2 (Feb. 2021), 227–247.

[27] BULLNHEIMER, B., HARTL, R., AND STRAUSS, C. An improved ant system algorithm for thevehicle routing problem. *Annals of Operations Research 89* (1999), 319–328.

[28] BURKE, E. K., GENDREAU, M., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E., AND QU, R. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society 64*, 12 (2013), 1695–1724.

[29] BURKE, E. K., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E., AND WOODWARD, J. R. A classification of hyper-heuristic approaches. In *Handbook of Metaheuristics*. Springer US, 2010, pp. 449–468.

[30] BURKE, E. K., HYDE, M. R., KENDALL, G., OCHOA, G., OZCAN, E., AND WOODWARD, J. R. Exploring hyper-heuristic methodologies with genetic programming. In *Computational intelligence*. Springer, 2009, pp. 177–201.

[31] BURKE, E. K., HYDE, M. R., KENDALL, G., OCHOA, G., ÖZCAN, E., AND WOODWARD, J. R. A classification of hyper-heuristic approaches: revisited. *Handbook of metaheuristics* (2019), 453–477.

[32] CAMPELO, P., NEVES-MOREIRA, F., AMORIM, P., AND ALMADA-LOBO, B. Consistent vehicle routing problem with service level agreements: A case study in the pharmaceutical distribution sector. *European Journal of Operational Research 273*, 1 (Feb. 2019), 131–145.

[33] CHEN, P., KUAN HUANG, H., AND DONG, X.-Y. Iterated variable neighborhood descent algorithm for the capacitated vehicle routing problem. *Expert Systems with Applications 37*, 2 (mar 2010), 1620–1627.

[34] CHRISTIAENS, J., AND VANDEN BERGHE, G. Slack induction by string removals for vehicle routing problems. *Transportation Science 54*, 2 (2020), 417–433.

[35] CHRISTOFIDES, N., AND EILON, S. An algorithm for the vehicle-dispatching problem. *Journal of the Operational Research Society 20*, 3 (1969), 309–318.

[36] CHRISTOFIDES, N., MINGOZZI, A., AND TOTH, P. State-space relaxation procedures for the computation of bounds to routing problems. *Networks 11*, 2 (1981), 145–164.

[37] CLARKE, G., AND WRIGHT, J. W. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research 12*, 4 (1964), 568–581.

[38] CORAL, D. B., SANTOS, M. O., TOLEDO, C., AND NIÑO, L. F. Clustering-based search in a memetic algorithm for the vehicle routing problem with time windows. In *2018 IEEE Congress on Evolutionary Computation (CEC)* (2018), IEEE, pp. 1–8.

[39] CORDEAU, J.-F., GENDREAU, M., HERTZ, A., LAPORTE, G., AND SORMANY, J.-S. New heuristics for the vehicle routing problem. In *Logistics Systems: Design and Optimization*. Springer-Verlag, 2005, pp. 279–297.

[40] CORDEAU, J.-F., LAPORTE, G., AND MERCIER, A. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society 52*, 8 (aug 2001), 928–936.

[41] COSTA, J. G. C. The vehicle routing problem with drones. 127.

[42] COSTA, J. G. C., MEI, Y., AND ZHANG, M. Adaptive search space through evolutionary hyper-heuristics for the large-scale vehicle routing problem. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)* (Dec. 2020), IEEE, pp. 2415–2422.

[43] COWLING, P., KENDALL, G., AND HAN, L. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. *Proceedings of the 2002 Congress on Evolutionary Computation. CEC-02 (Cat. No.02TH8600)* (2002).

[44] ČREPINŠEK, M., LIU, S.-H., AND MERNIK, M. Exploration and exploitation in evolutionary algorithms. *ACM Computing Surveys 45*, 3 (jun 2013), 1–33.

[45] CROES, G. A. A method for solving traveling-salesman problems. *Operations Research 6*, 6 (dec 1958), 791–812.

[46] DANTZIG, G. B., AND RAMSER, J. H. The truck dispatching problem. *Management Science 6*, 1 (oct 1959), 80–91.

[47] DE LA VEGA, J., MUNARI, P., AND MORABITO, R. Robust optimization for the vehicle routing problem with multiple deliverymen. *Central European Journal of Operations Research 27*, 4 (Dec. 2017), 905–936.

[48] DE LA VEGA, J., MUNARI, P., AND MORABITO, R. Exact approaches to the robust vehicle routing problem with time windows and multiple deliverymen. *Computers & Operations Research 124* (Dec. 2020), 105062.

[49] DESROSIERS, J., SOUMIS, F., AND DESROCHERS, M. Routing with time windows by column generation. *Networks 14*, 4 (1984), 545–565.

[50] DORIGO, M. Optimization, learning and natural algorithms [ph. d. thesis]. *Politecnico di Milano, Italy* (1992).

[51] DORIGO, M., AND DI CARO, G. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)* (1999), IEEE.

[52] DORIGO, M., AND STÜTZLE, T. Ant colony optimization: Overview and recent advances. In *Handbook of Metaheuristics*. Springer US, 2010, pp. 227–263.

[53] DORLING, K., HEINRICHS, J., MESSIER, G. G., AND MAGIEROWSKI, S. Vehicle routing problems for drone delivery. *IEEE Transactions on Systems, Man, and Cybernetics: Systems 47*, 1 (Jan. 2017), 70–85.

[54] DRAKE, J. H., KHEIRI, A., ÖZCAN, E., AND BURKE, E. K. Recent advances in selection hyper-heuristics. *European Journal of Operational Research 285*, 2 (2020), 405–428.

[55] DRAKE, J. H., KILILIS, N., AND ÖZCAN, E. Generation of VNS components with grammatical evolution for vehicle routing. In *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, pp. 25–36.

[56] DUECK, G. New optimization heuristics. *Journal of Computational Physics 104*, 1 (jan 1993), 86–92.

[57] EIBEN, A., AND SCHIPPERS, C. On evolutionary exploration and exploitation. *Fundamenta Informaticae 35*, 1-4 (1998), 35–50.

[58] EIBEN, A. E., AND SMITH, J. E. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2003.

[59] EILON, S., WATSON-GANDY, C. D. T., CHRISTOFIDES, N., AND DE NEUFVILLE, R. Distribution management-mathematical modelling and practical analysis. *IEEE Transactions on Systems, Man, and Cybernetics SMC-4*, 6 (nov 1974), 589–589.

[60] EWBANK, H., WANKE, P., AND HADI-VENCHEH, A. An unsupervised fuzzy clustering approach to the capacitated vehicle routing problem. 857–867.

[61] FAZI, S., FRANSOO, J. C., VAN WOENSEL, T., AND DONG, J.-X. A variant of the split vehicle routing problem with simultaneous deliveries and pickups for inland container shipping in dry-port based systems. *Transportation Research Part E: Logistics and Transportation Review 142* (Oct. 2020), 102057.

[62] FISHER, M. L., AND JAIKUMAR, R. A generalized assignment heuristic for vehicle routing. *Networks 11*, 2 (1981), 109–124.

[63] FLESZAR, K., OSMAN, I. H., AND HINDI, K. S. A variable neighbourhood search algorithm for the open vehicle routing problem. *European Journal of Operational Research 195*, 3 (jun 2009), 803–809.

[64] FLOOD, M. M. The traveling-salesman problem. *Operations Research 4*, 1 (feb 1956), 61–75.

[65] FRANCESCHI, R. D., FISCHETTI, M., AND TOTH, P. A new ILP-based refinement heuristic for vehicle routing problems. *Mathematical Programming 105*, 2-3 (nov 2005), 471–499.

[66] FUKASAWA, R., LONGO, H., LYSGAARD, J., DE ARAGÃO, M. P., REIS, M., UCHOA, E., AND WERNECK, R. F. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming 106*, 3 (oct 2005), 491–511.

[67] FUNKE, B., GRÜNERT, T., AND IRNICH, S. Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of Heuristics 11*, 4 (jul 2005), 267–306.

[68] GARRIDO, P., AND CASTRO, C. Stable solving of CVRPs using hyperheuristics. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (2009), ACM, pp. 255–262.

[69] GENDREAU, M., HERTZ, A., AND LAPORTE, G. A tabu search heuristic for the vehicle routing problem. *Management Science 40*, 10 (oct 1994), 1276–1290.

[70] GENDREAU, M., LAPORTE, G., AND POTVIN, J.-Y. 6. metaheuristics for the capacitated VRP. In *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, jan 2002, pp. 129–154.

[71] GENDREAU, M., AND POTVIN, J.-Y. *Handbook of Metaheuristics*. Springer-Verlag New York Inc., 2010.

[72] GENDREAU, M., AND POTVIN, J.-Y. Tabu search. In *Handbook of Metaheuristics*. Springer US, 2010, pp. 41–59.

[73] GENDREAU, M., AND POTVIN, J.-Y., Eds. *Handbook of Metaheuristics*. Springer International Publishing, 2019.

[74] GENDREAU, M., POTVIN, J.-Y., BRÄYSY, O., HASLE, G., AND LØKETANGEN, A. Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography. In *Operations Research/Computer Science Interfaces*. Springer US, 2008, pp. 143–169.

[75] GENDREAU, M., AND TARANTILIS, C. D. *Solving large-scale vehicle routing problems with time windows: The state-of-the-art*. Cirrelt Montreal, 2010.

[76] GILLETT, B. E., AND MILLER, L. R. A heuristic algorithm for the vehicle-dispatch problem. *Operations research 22*, 2 (1974), 340–349.

[77] GLOVER, F. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics 65*, 1-3 (mar 1996), 223–253.

[78] GLOVER, F. Tabu search and adaptive memory programming - advances, applications and challenges. In *Operations Research/Computer Science Interfaces Series*. Springer US, 1997, pp. 1–75.

[79] GLOVER, F. W. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res. 13*, 5 (1986), 533–549.

[80] GOLDEN, B. L., WASIL, E. A., KELLY, J. P., AND CHAO, I.-M. The impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets, and computational results. *Fleet Management and Logistics* (1998), 33–56.

[81] GOMES, L., AND VON ZUBEN, F. A heuristic method based on unsupervised learning and fuzzy inference for the vehicle routing problem. In *VII Brazilian Symposium on Neural Networks, 2002. SBRN 2002. Proceedings.* (2002), pp. 130–135.

[82] GROËR, C., GOLDEN, B., AND WASIL, E. A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation 2*, 2 (2010), 79–101.

[83] GULIC, M., AND JAKOBOVIC, D. Evolution of vehicle routing problem heuristics with genetic programming. In *2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (2013), IEEE, pp. 988–992.

[84] HAN, H., AND CUETO, E. P. Waste collection vehicle routing problem: Literature review. *PROMET - Traffic&Transportation 27*, 4 (Aug. 2015), 345–358.

[85] HAN, L., KENDALL, G., AND COWLING, P. An adaptive length chromosome hyper-heuristic genetic algorithm for a trainer scheduling problem. In *Recent Advances in Simulated Evolution and Learning*. World Scientific, 2004, pp. 506–525.

[86] HANSEN, P., MLADENOVIĆ, N., BRIMBERG, J., AND PÉREZ, J. A. M. Variable neighborhood search. In *Handbook of Metaheuristics*. Springer US, 2010, pp. 61–86.

[87] HASLE, G., AND KLOSTER, O. Industrial vehicle routing. *Geometric Modelling, Numerical Simulation, and Optimization* (2007), 397–435.

[88] HE, R., XU, W., SUN, J., AND ZU, B. Balanced k-means algorithm for partitioning areas in large-scale vehicle routing problem. In *2009 Third International Symposium on Intelligent Information Technology Application* (Nov. 2009), vol. 3, IEEE, pp. 87–90.

[89] HELD, M., AND KARP, R. M. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming 1*, 1 (dec 1971), 6–25.

[90] HELSGAUN, K. An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational Research 126*, 1 (oct 2000), 106–130.

[91] HELSGAUN, K. An extension of the lin-kernighan-helsgaun TSP solver for constrained traveling salesman and vehicle routing problems.

[92] HO, S. C., AND GENDREAU, M. Path relinking for the vehicle routing problem. *Journal of Heuristics 12*, 1-2 (mar 2006), 55–72.

[93] HOLLAND, J. *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, Ann Arbor, 1975.

[94] HOTTUNG, A., AND TIERNEY, K. Neural large neighborhood search for routing problems. 103786.

[95] HUANG, M., AND HU, X. Large scale vehicle routing problem: An overview of algorithms and an intelligent procedure. *International Journal of Innovative Computing, Information and Control 8*, 8 (2012), 5809–5819.

[96] J. R. ARAQUE, G., KUDVA, G., MORIN, T. L., AND PEKNY, J. F. A branch-and-cut algorithm for vehicle routing problems. *Annals of Operations Research 50*, 1 (dec 1994), 37–59.

[97] JACOBSEN-GROCOTT, J., MEI, Y., CHEN, G., AND ZHANG, M. Evolving heuristics for dynamic vehicle routing with time windows using genetic programming. In *2017 IEEE Congress on Evolutionary Computation (CEC)* (2017), IEEE, pp. 1948–1955.

[98] JAIN, A. K. Data clustering: 50 years beyond k-means. *Pattern recognition letters 31*, 8 (2010), 651–666.

[99] JI, P., AND CHEN, K. The vehicle routing problem: The case of the hong kong postal service. *Transportation Planning and Technology 30*, 2–3 (Apr. 2007), 167–182.

[100] JIANG, G., DONG, H., YANG, L., LI, G., AND XIANG, F. Hyperheuristic genetic algorithm for steelmaking continuous casting rescheduling based on strong disturbance of task. *International Journal of Wireless and Mobile Computing 15*, 3 (2018), 231.

[101] KENNEDY, J. Swarm intelligence. In *Handbook of Nature-Inspired and Innovative Computing*. Kluwer Academic Publishers, 2006, pp. 187–219.

[102] KIM, H., YANG, J., AND LEE, K.-D. Vehicle routing in reverse logistics for recycling end-of-life consumer electronic goods in south

korea. *Transportation Research Part D: Transport and Environment 14*, 5 (July 2009), 291–299.

[103] KINDERVATER, G. A. P., AND SAVELSBERGH, M. Vehicle routing : handling edge exchanges. In *Local Search in Combinatorial Optimization*. Princeton University Press, 1997.

[104] KIRKPATRICK, S., GELATT, C. D., AND VECCHI, M. P. Optimization by simulated annealing. *Science 220*, 4598 (may 1983), 671–680.

[105] KLEE, V., AND MINTY, G. J. How good is the simplex algorithm. *Inequalities 3*, 3 (1972), 159–175.

[106] KONSTANTAKOPOULOS, G. D., GAYIALIS, S. P., AND KECHAGIAS, E. P. Vehicle routing problem and related algorithms for logistics distribution: a literature review and classification. *Operational Research 22*, 3 (Sept. 2020), 2033–2062.

[107] KYTÖJOKI, J., NUORTIO, T., BRÄYSY, O., AND GENDREAU, M. An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research 34*, 9 (sep 2007), 2743–2757.

[108] LAN, W., YE, Z., RUAN, P., LIU, J., YANG, P., AND YAO, X. Region-focused memetic algorithms with smart initialization for real-world large-scale waste collection problems. *IEEE Transactions on Evolutionary Computation 26*, 4 (Aug. 2022), 704–718.

[109] LANGDON, W. B., MCKAY, R. I., AND SPECTOR, L. Genetic programming. In *Handbook of metaheuristics*. Springer, 2010, pp. 185–225.

[110] LAPORTE, G. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research 59*, 3 (1992), 345–358.

[111] LAPORTE, G. Fifty years of vehicle routing. *Transportation Science 43*, 4 (nov 2009), 408–416.

[112] LAPORTE, G. A concise guide to the traveling salesman problem. *Journal of the Operational Research Society 61*, 1 (jan 2010), 35–40.

[113] LAPORTE, G., AND NOBERT, Y. A branch and bound algorithm for the capacitated vehicle routing problem. *Operations-Research-Spektrum 5*, 2 (1983), 77–85.

[114] LAPORTE, G., NOBERT, Y., AND DESROCHERS, M. Optimal routing under capacity and distance restrictions. *Operations research 33*, 5 (1985), 1050–1073.

[115] LAPORTE, G., AND SEMET, F. 5. classical heuristics for the capacitated VRP. In *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, jan 2002, pp. 109–128.

[116] LAWLER, E. L., AND WOOD, D. E. Branch-and-bound methods: A survey. *Operations research 14*, 4 (1966), 699–719.

[117] LENSTRA, J. K., AND KAN, A. H. G. R. Complexity of vehicle routing and scheduling problems. *Networks 11*, 2 (1981), 221–227.

[118] LI, F., GOLDEN, B., AND WASIL, E. Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research 32*, 5 (may 2005), 1165–1179.

[119] LIN, S., AND KERNIGHAN, B. W. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research 21*, 2 (apr 1973), 498–516.

[120] LING CHEN, A., KE YANG, G., AND MING WU, Z. Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem. *Journal of Zhejiang University-SCIENCE A 7*, 4 (mar 2006), 607–614.

[121] LIU, H., ZHANG, Z., AND GUO, X. Restricted neighborhood search for large scale vehicle routing problems. In *2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC)* (may 2019), IEEE.

[122] LIU, Y., MEI, Y., ZHANG, M., AND ZHANG, Z. Automated heuristic design using genetic programming hyper-heuristic for uncertain capacitated arc routing problem. In *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2017), GECCO -17, Association for Computing Machinery, pp. 290–297.

[123] LÓPEZ-IBÁÑEZ, M., DUBOIS-LACOSTE, J., PÉREZ CÁCERES, L., BIRATTARI, M., AND STÜTZLE, T. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives 3* (2016), 43–58.

[124] LOURENÇO, H. R., MARTIN, O. C., AND STÜTZLE, T. Iterated local search: Framework and applications. In *Handbook of Metaheuristics*. Springer US, 2010, pp. 363–397.

[125] MACHADO, A. M., MAURI, G. R., BOERES, M. C. S., AND DE ROSA, R. A. A new hybrid matheuristic of grasp and vns based on constructive heuristics, set-covering and set-partitioning formulations applied to the capacitated vehicle routing problem. *Expert Systems with Applications 184* (Dec. 2021), 115556.

[126] MACQUEEN, J., ET AL. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (1967), no. 14, Oakland, CA, USA, pp. 281–297.

[127] MARSHALL, R. J., JOHNSTON, M., AND ZHANG, M. Developing a hyper-heuristic using grammatical evolution and the capacitated

vehicle routing problem. In *SEAL* (2014), vol. 8886 of *Lecture Notes in Computer Science*, Springer, pp. 668–679.

[128] MARSHALL, R. J., JOHNSTON, M., AND ZHANG, M. Hyper-heuristics, grammatical evolution and the capacitated vehicle routing problem. In *Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion - GECCO Comp'14* (2014), ACM Press.

[129] MEI, Y., AND ZHANG, M. Genetic programming hyper-heuristic for stochastic team orienteering problem with time windows. In *2018 IEEE Congress on Evolutionary Computation (CEC)* (July 2018), IEEE, pp. 1–8.

[130] MEIGNAN, D., KOUKAM, A., AND CRÉPUT, J.-C. Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics 16*, 6 (dec 2009), 859–879.

[131] MESTER, D., AND BRÄYSY, O. Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Computers & Operations Research 34*, 10 (oct 2007), 2964–2975.

[132] MICHALEWICZ, Z. *Genetic algorithms + data structures = evolution programs (3nd, extended ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[133] MITRA, S. A parallel clustering technique for the vehicle routing problem with split deliveries and pickups. *Journal of the operational Research Society 59*, 11 (2008), 1532–1546.

[134] MLADENOVIC, N., AND HANSEN, P. Variable neighborhood search. *Computers & Operations Research 24*, 11 (nov 1997), 1097–1100.

[135] MOLE, R. H., AND JAMESON, S. R. A sequential route-building algorithm employing a generalised savings criterion. *Journal of the Operational Research Society 27*, 2 (jun 1976), 503–511.

[136] MOUSTAFA, A., ABDELHALIM, A. A., ELTAWIL, A. B., AND FORS, N. Waste collection vehicle routing problem: Case study in alexandria, egypt. *The 19th International Conference on Industrial Engineering and Engineering Management* (2013), 935–944.

[137] MUÑOZ-HERRERA, S., AND SUCHAN, K. Constrained fitness landscape analysis of capacitated vehicle routing problems. 53. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[138] NAZARI, M., OROOJLOOY, A., SNYDER, L., AND TAKAC, M. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems* (2018), S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc.

[139] NEWTON, R. M., AND THOMAS, W. H. Bus routing in a multi-school system. *Computers & Operations Research 1*, 2 (aug 1974), 213–222.

[140] NIKOLAEV, A. G., AND JACOBSON, S. H. Simulated annealing. In *Handbook of Metaheuristics*. Springer US, 2010, pp. 1–39.

[141] OR, I. *TRAVELING SALESMAN TYPE COMBINATORIAL PROBLEMS AND THEIR RELATION TO THE LOGISTICS OF REGIONAL BLOOD BANKING.* PhD thesis, Northwestern University, 1977.

[142] OSMAN, I. H. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research 41*, 4 (dec 1993), 421–451.

[143] PAPADIMITRIOU, C. H., AND STEIGLITZ, K. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[144] PECIN, D., PESSOA, A., POGGI, M., AND UCHOA, E. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation 9*, 1 (jun 2016), 61–100.

[145] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., AND DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[146] PESSOA, A., SADYKOV, R., UCHOA, E., AND VANDERBECK, F. A generic exact solver for vehicle routing and related problems. In *Integer Programming and Combinatorial Optimization*. Springer International Publishing, 2019, pp. 354–369.

[147] PESSOA, A., SADYKOV, R., UCHOA, E., AND VANDERBECK, F. A generic exact solver for vehicle routing and related problems. *Mathematical Programming 183*, 1–2 (June 2020), 483–523.

[148] PILLAY, N. Evolving hyper-heuristics for the uncapacitated examination timetabling problem. *Journal of the Operational Research Society 63*, 1 (Jan. 2012), 47–58.

[149] PILLAY, N. A study of evolutionary algorithm selection hyper-heuristics for the one-dimensional bin-packing problem. *South African Computer Journal 48* (2012), 31–40.

[150] PILLAY, N., AND QU, R. Vehicle routing problems. In *Natural Computing Series*. Springer International Publishing, 2018, pp. 51–60.

[151] PISINGER, D., AND ROPKE, S. A general heuristic for vehicle routing problems. *Computers & Operations Research 34*, 8 (aug 2007), 2403–2435.

[152] POGGI, M., AND UCHOA, E. New exact algorithms for the capacitated vehicle routing problem. In *Vehicle Routing: Problems, Methods, and Applications*, P. Toth and D. Vigo, Eds. SIAM, 2014, pp. 59–86.

[153] POTVIN, J.-Y. State-of-the art review—evolutionary algorithms for vehicle routing. *INFORMS Journal on Computing 21*, 4 (nov 2009), 518–548.

[154] POTVIN, J.-Y., AND ROUSSEAU, J.-M. An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society 46*, 12 (dec 1995), 1433–1446.

[155] PRINS, C. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research 31*, 12 (oct 2004), 1985–2002.

[156] QU, R., KENDALL, G., AND PILLAY, N. The general combinatorial optimization problem: Towards automated algorithm design. *IEEE Computational Intelligence Magazine 15*, 2 (2020), 14–23.

[157] QUEIROGA, E., SADYKOV, R., AND UCHOA, E. A popmusic matheuristic for the capacitated vehicle routing problem. *Computers & Operations Research 136* (Dec. 2021), 105475.

[158] RAGHAVENDRA, A., KRISHNAKUMAR, T., MURALIDHAR, R., SARVANAN, D., AND RAGHAVENDRA, B. A practical heuristic for a large scale vehicle routing problem. *European Journal of Operational Research 57*, 1 (feb 1992), 32–38.

[159] RAGHAVJEE, R., AND PILLAY, N. A genetic algorithm selection perturbative hyper-heuristic for solving the school timetabling problem. *ORiON 31*, 1 (2015), 39–60.

[160] RALPHS, T., KOPMAN, L., PULLEYBLANK, W., AND TROTTER, L. On the capacitated vehicle routing problem. *Mathematical Programming 94*, 2-3 (jan 2003), 343–359.

[161] REEVES, C. R. Genetic algorithms. In *Handbook of Metaheuristics*. Springer US, 2010, pp. 109–139.

[162] REIMANN, M., DOERNER, K., AND HARTL, R. F. D-ants: Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research 31*, 4 (apr 2004), 563–591.

[163] RENAUD, J., LAPORTE, G., AND BOCTOR, F. F. A tabu search heuristic for the multi-depot vehicle routing problem. *Computers & Operations Research 23*, 3 (Mar. 1996), 229–235.

[164] RESENDE, M. G., RIBEIRO, C. C., GLOVER, F., AND MARTÍ, R. Scatter search and path-relinking: Fundamentals, advances, and applications. In *Handbook of Metaheuristics*. Springer US, 2010, pp. 87–107.

[165] ROSS, P. Hyper-heuristics. In *Search Methodologies*. Springer US, jul 2013, pp. 611–638.

[166] RUSSELL, S., AND NORVIG, P. *Artificial Intelligence: A Modern Approach, Global Edition*. Addison Wesley, 2018.

[167] SABAR, N. R., ZHANG, X. J., AND SONG, A. A math-hyper-heuristic approach for large-scale vehicle routing problems with time windows. In *CEC* (2015), IEEE, pp. 830–837.

[168] SANCHEZ, M., CRUZ-DUARTE, J. M., ORTIZ-BAYLISS, J. C., CEBALLOS, H., TERASHIMA-MARIN, H., AND AMAYA, I. A systematic review of hyper-heuristics on combinatorial optimization problems. *IEEE Access 8* (2020), 128068–128095.

[169] SAVELSBERGH, M. W. Local search in routing problems with time windows. *Annals of Operations research 4*, 1 (1985), 285–305.

[170] SCHRIMPF, G., SCHNEIDER, J., STAMM-WILBRANDT, H., AND DUECK, G. Record breaking optimization results using the ruin and

recreate principle. *Journal of Computational Physics 159*, 2 (apr 2000), 139–171.

[171] SHAW, P. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming - CP98*. Springer Berlin Heidelberg, 1998, pp. 417–431.

[172] SIM, K., AND HART, E. A combined generative and selective hyper-heuristic for the vehicle routing problem. In *GECCO* (2016), T. Friedrich, F. Neumann, and A. M. Sutton, Eds., ACM, pp. 1093–1100.

[173] SMITH-MILES, K. A. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys 41*, 1 (Jan. 2009), 1–25.

[174] SONG, B. D., AND KO, Y. D. A vehicle routing problem of both refrigerated- and general-type vehicles for perishable food products delivery. *Journal of Food Engineering 169* (Jan. 2016), 61–71.

[175] SPEARS, W. M., JONG, K. A., BÄCK, T., FOGEL, D. B., AND GARIS, H. An overview of evolutionary computation. *Lecture Notes in Computer Science* (1993), 442–459.

[176] SUBRAMANIAN, A., UCHOA, E., AND OCHI, L. S. A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research 40*, 10 (oct 2013), 2519–2531.

[177] SYRICHAS, A., AND CRISPIN, A. Large-scale vehicle routing problems: Quantum annealing, tunings and results. *Computers & Operations Research 87* (nov 2017), 52–62.

[178] TAILLARD, É., BADEAU, P., GENDREAU, M., GUERTIN, F., AND POTVIN, J.-Y. A tabu search heuristic for the vehicle routing prob-

lem with soft time windows. *Transportation Science 31*, 2 (may 1997), 170–186.

[179] THOMPSON, P. M., AND PSARAFTIS, H. N. Cyclic transfer algorithm for multivehicle routing and scheduling problems. *Operations Research 41*, 5 (oct 1993), 935–946.

[180] TOTH, P., AND TRAMONTANI, A. An integer linear programming local search for capacitated vehicle routing problems. In *Operations Research/Computer Science Interfaces*. Springer US, 2008, pp. 275–295.

[181] TOTH, P., AND VIGO, D. *The vehicle routing problem*. Society for Industrial and Applied Mathematics, Philadelphia, 2002.

[182] TU, W., LI, Q., LI, Q., ZHU, J., ZHOU, B., AND CHEN, B. A spatial parallel heuristic approach for solving very large-scale vehicle routing problems. *Transactions in GIS 21*, 6 (mar 2017), 1130–1147.

[183] UCHOA, E., PECIN, D., PESSOA, A. A., POGGI, M., VIDAL, T., AND SUBRAMANIAN, A. New benchmark instances for the capacitated vehicle routing problem. *Eur. J. Oper. Res. 257*, 3 (2017), 845–858.

[184] VIDAL, T. Hybrid genetic search for the CVRP: Open-source implementation and SWAP$*$ neighborhood. *Computers & Operations Research 140* (apr 2022), 105643.

[185] VIDAL, T., CRAINIC, T. G., GENDREAU, M., LAHRICHI, N., AND REI, W. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research 60*, 3 (jun 2012), 611–624.

[186] VIDAL, T., CRAINIC, T. G., GENDREAU, M., AND PRINS, C. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research 231*, 1 (nov 2013), 1–21.

[187] VIDAL, T., CRAINIC, T. G., GENDREAU, M., AND PRINS, C. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research 234*, 3 (may 2014), 658–673.

[188] VIEIRA, Y. E. M., DE BANDEIRA, R. A. M., AND DA SILVA JÚNIOR, O. S. Multi-depot vehicle routing problem for large scale disaster relief in drought scenarios: The case of the brazilian northeast region. *International Journal of Disaster Risk Reduction 58* (May 2021), 102193.

[189] VOUDOURIS, C., AND TSANG, E. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research 113*, 2 (mar 1999), 469–499.

[190] VOUDOURIS, C., TSANG, E. P., AND ALSHEDDY, A. Guided local search. In *Handbook of Metaheuristics*. Springer US, 2010, pp. 321–361.

[191] WANG, Q., AND TANG, C. Deep reinforcement learning for transportation network combinatorial optimization: A survey. *Knowledge-Based Systems 233* (2021), 107526.

[192] WANG, S., LIU, F., LIAN, L., HONG, Y., AND CHEN, H. Integrated post-disaster medical assistance team scheduling and relief supply distribution. *The International Journal of Logistics Management 29*, 4 (June 2018), 1279–1305.

[193] WANG, S., MEI, Y., AND ZHANG, M. A multi-objective genetic programming hyper-heuristic approach to uncertain capacitated arc routing problems. In *CEC* (2020), IEEE, pp. 1–8.

[194] WOLPERT, D. H., AND MACREADY, W. G. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation 1*, 1 (1997), 67–82.

[195] WREN, A., AND HOLLIDAY, A. Computer scheduling of vehicles from one or more depots to a number of delivery points. *Journal of the Operational Research Society 23*, 3 (sep 1972), 333–344.

[196] XIAO, J., ZHANG, T., DU, J., AND ZHANG, X. An evolutionary multiobjective route grouping-based heuristic algorithm for large-scale capacitated vehicle routing problems. *IEEE transactions on cybernetics* (2019), 1–14.

[197] YELMEWAD, P., AND TALAWAR, B. Parallel version of local search heuristic algorithm to solve capacitated vehicle routing problem. *Cluster Computing 24*, 4 (July 2021), 3671–3692.

[198] YU, B., YANG, Z.-Z., AND YAO, B. An improved ant colony optimization for vehicle routing problem. *European Journal of Operational Research 196*, 1 (jul 2009), 171–176.

[199] YÜCENUR, G. N., AND DEMIREL, N. Ç. A new geometric shape-based genetic clustering algorithm for the multi-depot vehicle routing problem. *Expert Systems with Applications 38*, 9 (2011), 11859–11865.

[200] ZHANG, F., MEI, Y., NGUYEN, S., AND ZHANG, M. Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job shop scheduling. *IEEE Transactions on Cybernetics* (2020.).

[201] ZHANG, F., MEI, Y., AND ZHANG, M. A two-stage genetic programming hyper-heuristic approach with feature selection for dynamic flexible job shop scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2019), pp. 347–355.

[202] ZHUO, E., DENG, Y., SU, Z., YANG, P., YUAN, B., AND YAO, X. An experimental study of large-scale capacitated vehicle routing problems, 2019.

[203] ŽUNIĆ, E., DONKO, D., AND BUZA, E. An adaptive data-driven approach to solve real-world vehicle routing problems in logistics. 1–24.