GENETIC ALGORITHM APPLICATIONS FOR THE VEHICLE ROUTING PROBLEM WITH ROAMING DELIVERY LOCATIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

By
Serkan Turhan
June 2021

Genetic Algorithm Applications for the Vehicle Routing Problem with Roaming Delivery Locations
By Serkan Turhan
June 2021

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Bahar Yetiş(Advisor)

Oya Karaşan

Amine Gizem Tiniç

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

GENETIC ALGORITHM APPLICATIONS FOR THE VEHICLE ROUTING PROBLEM WITH ROAMING DELIVERY LOCATIONS

Serkan Turhan
M.S. in Industrial Engineering
Advisor: Bahar Yetiş
June 2021

The recent innovations in the e-commerce industry developed a new delivery option where the orders of the customers can be delivered to the trunks of their cars. Compared to the conventional home-delivery, this option is not only able to decrease the total distance traveled but also increase the customer satisfaction by decreasing the number of failed deliveries. The problem introduced by this option is called the vehicle routing problem with roaming delivery locations. This thesis proposes a new, time-efficient solution construction strategy for the problem. The construction strategy is able to represent any feasible solution for the problem and has a complexity linearly increasing with the number of delivery nodes in the problem. Based on the constructor, a new genetic algorithm to find and improve solutions to near-optimal within polynomial time is proposed. Furthermore, a separate, new fine-tuning algorithm to improve the parameters of the genetic algorithm for a given set of problem instances is proposed. The most notable feature of the proposed genetic algorithm is the time-efficiency as it is able to construct a solution within milliseconds for the largest problem instance available in the literature and the computation time scales with the problem size linearly. Parallel computing can be implemented in both the fine-tuning and the genetic algorithm, which allows better results in a shorter processing time. Within 5 minutes of computation time, the fine-tuned genetic algorithm found optimal solutions in 8 out of 19 instances with known optimal solutions, moreover, it was able to find solutions better than the previous solution methodologies in 12 out of 60 instances used in our experiments. Gaps between the results of the proposed genetic algorithm and the best solution found by a commercial solver (CPLEX) are between (0.0%, 26.2%) and (-6.0%, 16.3%) in small-medium and large instances respectively.

Keywords: vehicle routing problem with roaming delivery locations, genetic algorithm, evolutionary algorithm, generalized vehicle routing with time windows.

ÖZET

HAREKETLI TESLIMAT NOKTALI ARAÇ ROTALAMA PROBLEMLERI IÇIN GELIŞTIRILMIŞ GENETIK ALGORITMALAR

Serkan Turhan
Endüstri Mühendisliği, Yüksek Lisans
Tez Danışmanı: Bahar Yetiş
Haziran 2021

E-ticaret endüstrisindeki son yenilikler ile müşterilerin siparişlerinin arabalarının bagajına teslim edilebildiği yeni bir teslimat seçeneği geliştirdi. Sıkça kullanılan eve yapılan teslimatlar ile karşılaştırıldığında, bu seçenek sadece kat edilen toplam mesafeyi azaltmakla kalmıyor ve başarısız teslimatları azaltarak müşteri memnuniyetini de arttırıyor. Bu seçenek ile ortaya çıkan probleme hareketli teslimat noktalı araç rotalama problemi denir. Bu tez, problem için yeni ve zaman açısından verimli bir çözüm oluşturma stratejisi önermektedir. Bu strateji problem değişkenlerinden bağımsız şekilde problem için her uygun çözümü temsil edebilir. Bu stratejiyi temel alan, polinom zamanında en iyi çözümlere yakın çözümler bulmak ve geliştirmek için yeni bir genetik algoritma önerilmiştir. Ayrıca, belirli bir dizi problem örneği için, genetik algoritmanın değişkenlerini iyileştirmek için farklı, yeni bir ince ayar algoritması önerilmiştir. Önerilen genetik algoritmanın en önemli özelliği, çözüm oluşturma zamanının problem boyutu ile doğrusal ölçekte artması ve literatürde mevcut olan en büyük problem örneği için milisaniyeler içinde bir çözüm oluşturabilmesidir. Paralel hesaplama hem ince ayar algoritmasında hem de genetik algoritmalarda uygulanabilir, bu da zamanın daha verimli kullanılmasını ve daha sürede daha iyi sonuçlar alınmasını sağlar. 5 dakikalık işlem süresi içinde, ince ayarlı genetik algoritma, 19 örneğin 8'inde en iyi çözümleri buldu, ayrıca 60 örnekten 12'sinde önceki çözüm metodolojilerinden daha iyi çözümler bulabildi. Önerilen genetik algoritmanın sonuçları ile çözücü tarafından bulunan en iyi çözüm arasındaki farklar, küçük-orta ve büyük örneklerde sırasıyla (%0.0, %26.2) ve (-%6.0, %16.3) arasındadır.

Anahtar sözcükler: hareketli teslimat noktalı araç rotalama problemi, genetik algoritma, evrimsel algoritma, genelleştirilmiş zaman bağımlı araç rotalama problemi.

Acknowledgement

First of all, I would like to express my gratitude for my advisor Prof. Bahar Yetiş Kara, without whom I wouldn't even consider this degree and miss-out on this great opportunity. Her continuous support, knowledge and experience have not only improved my research and academic skills but also motivated me to develop myself in and apply my skills to different areas. It has been a great pleasure to work under her guidance, both during my undergraduate senior project and master's thesis.

I'm also grateful to Prof. Oya Karaşan and Prof. Amine Gizem Tiniç for their support and guidance in the development of this thesis and accepting to read and review it, providing invaluable comments and suggestions.

Additionally, I would like to thank Mr. Kadri Yetiş for his support in server-related issues, during my studies, some of the algorithms had to run for a couple months and, thanks to him, there wasn't any technical difficulty. I also would like to thank the Industrial Engineering Department and my friends Beste Akbaş, Nilsu Uzunlar, Mahsa Nakhost and office 305 for having such a friendly and supportive environment.

Finally, I would like to thank my parents Hüseyin and Ayla Turhan, and my brother Kerem. Their support and understanding during my studies means a lot to me and words fail to express my gratitude. I'm and will always be grateful for their support, guidance and sacrifice.

Contents

1 Introduction **1**

2 Problem Definition **4**

2.1 Emergence of the Trunk Delivery 4

2.2 Vehicle Routing Problem with Roaming Delivery Locations 6

2.2.1 Relation with Generalized VRP 6

2.2.2 Relation with VRP with Time Windows 7

3 Literature Review **9**

3.1 VRP with Roaming Delivery Locations 9

3.1.1 Stochastic Perspective 11

3.2 Genetic Algorithm 12

3.2.1 Description 12

3.2.2 Advantages 13

3.2.3 Comparison with other heuristics 15

3.2.4 Applications for VRP-TW and GVRP 16

4 Mathematical Model 18

4.1 Operational Dynamics and Related Assumptions 19

5 Solution Methodology 21

5.1 Genetic Algorithm 21

5.1.1 General Structure 22

5.1.2 Solution Representation and Fitness Function 23

5.1.3 Initial Population 24

5.1.4 Constructing the Next Population 24

5.2 Solver GA (SGA) 27

5.2.1 Solution Representation and Fitness Function 27

5.2.2 Initial Population 31

5.2.3 Algorithm Setup 32

5.2.4 Relaxing the Assumptions 36

5.3 Fine-tuning Algorithm 39

5.3.1 Chromosome Structure and Initial Population 40

5.3.2 Constructing the Next Population 42

5.3.3 Terminating Condition 45

6 Results **46**

6.1 Fine-tuning Algorithm Results 46

6.2 Solver-GA Results 47

6.2.1 Removal of the Infeasible Nodes 47

6.2.2 Mathematical Model and Solver-GA 48

7 Conclusions and Future Research Directions **52**

A Benchmark Instances **64**

B Solver-GA Results **66**

C Solver-GA Solution Results **69**

List of Figures

2.1 Example for a Generalized VRP setup 7

2.2 Example solution for Generalized VRP 7

2.3 Example GVRP-TW problem 8

2.4 Example VRP-RDL problem with customer itinerary arcs 8

3.1 Elements of binary matrix needed to be inverted to perform a swap of two nodes 13

3.2 Elements of binary matrix needed to be swapped to perform a swap of two nodes 14

3.3 Elements of vector representation needed to be swapped/changed to perform a swap of two nodes 14

5.1 Flowchart of GA 22

5.2 Example of chromosome and gene on an encoded solution 23

5.3 Example solution vector and resulting routes 28

5.4 Example solution vector and constructed routes 29

5.5 Partially matched crossover to produce the offspring from a parent pair 33

5.6 Order assignment with no split deliveries 36

5.7 Order assignment with split deliveries 36

5.8 Example of an inefficient route with duplicated customers 37

6.1 Fitness values over the generations 47

6.2 Gap vs. Node size distribution for 5 and 15 min. SGA 49

List of Tables

3.1 Literature review on VRP-TW with GA 17

4.1 Modeling parameters and decision variables 18

5.1 Commonly used selectors 25

5.2 Commonly used terminating conditions 27

5.3 Solver-GA parameters 32

5.4 Parameters of Fine-tuning GA 40

6.1 Summary of algorithm performance by number of customers (5 minutes) 50

6.2 Summary of algorithm performance by number of customers (15 minutes) 50

6.3 Summary of algorithm performance by number of nodes (5 minutes) 51

6.4 Summary of algorithm performance by number of nodes (15 minutes) 51

A.1 Benchmark Instances 64

A.2 Benchmark Instances continued 65

B.1 Model, BP and SGA results 66

B.2 Model, BP and SGA results continued 67

B.3 Model, BP and SGA results continued 68

C.1 Solution and objective comparisons 69

C.2 Solution and objective comparisons continued 70

C.3 Solution and objective comparisons continued 71

Chapter 1

Introduction

Logistics is one of the most important industries in the world, making up 10% of the EU's GDP [1]. Under the general logistics theme, city logistics is defined as the transportation activities both made in and made for the urban environment, and constitutes as a major enabling factor for the most economic and social activities [2]. One of the most common problems of the city logistics is the Vehicle Routing Problem (VRP), which deals with designing optimal routes emerging from one or many depots to a number of customers or cities, subject to system-specific constraints [3].

In 2017, DHL introduced in-car deliveries to its customers with compatible cars [4]. This delivery option enabled them to deliver the parcels of customers to their parked cars, without requiring the customers to be near their cars, as they travel throughout the day. In-car deliveries, also known as trunk deliveries, have two major benefits over the home delivery: prevention of failed deliveries due to no one being at home at the time of delivery and less distance traveled per delivery [5]. As the demand for this option continues to grow with the e-commerce companies, combined with the changing lifestyle of the customers, these benefits become more relevant [6].

The problem introduced by the option of in-car deliveries is called VRP with

Roaming Delivery Locations (VRP-RDL) [5], the itineraries of customers are assumed to be known, so the problem consists of finding routes that visit every customer at exactly one point throughout their itineraries, while minimizing the distance covered.

This thesis proposes a flexible, time-effective solution construction procedure that can be utilized to solve the VRP-RDL and its variants with necessary modifications. The proposed procedure is used with a meta-heuristic solution methodology to construct and improve a near-optimal solution. Finally, a fine-tuning methodology to improve the solution quality of the meta-heuristic for a given set of problem instances is proposed.

In Chapter 2, the emergence of VRP-RDL and its economical benefits are discussed. Amazon's in-car delivery feature is provided as the real-life application example. The structure and key characteristics of VRP-RDL, its possible variants in the real-life applications as well as its major differences from the similar problems in the literature are provided.

In Chapter 3, a literature review on VRP-RDL, its variants, the problems obtained by generalizing some aspects of it and the proposed solution methodologies are provided. It includes the exact and heuristic solution methodologies developed in the literature.

In Chapter 4, the exact solution methodology, with which the performance of the proposed solution methodology is compared, is provided. A mixed-integer formulation as well as the parameters of the problem are defined.

In Chapter 5, first the proposed solution construction procedure for the problem is provided. Second, a meta-heuristic that employs the procedure to find solutions competitive to the ones obtained using the mathematical model is provided. Finally, a fine-tuning procedure to improve the parameters of the meta-heuristic to obtain solutions of better quality for a given set of problem instances is provided.

In Chapter 6, the results obtained using the proposed solution methodology

on the problem is presented and compared with the existing methodologies in the literature.

In Chapter 7, the thesis is concluded with an overview of the work and a discussion about the potential future research directions.

Chapter 2

Problem Definition

In this chapter, the emergence of the VRP-RDL, its relation with the e-commerce, details about its real-life application as well as the stakeholder companies overseeing and improving its application are provided in detail.

2.1 Emergence of the Trunk Delivery

In the recent years, e-commerce started taking the place of retail stores, and it was noted that online sales was higher than the store sales in the US even before the corona virus outbreak [7]. The pandemic even furthered this outbreak in e-commerce [8]. In New York City alone, there are more than 1.5 million packages delivered each day [9], and this number is expected to grow with the increasing trend of e-commerce, not only in New York City but also world-wide [10]. One of the major stakeholders behind this growth is Amazon [7]. Founded in 1994 as an online bookstore, Amazon grew rapidly and was selling consumer electronics, video games, software, home-improvement items, toys and games by 1999, to over one million customers [11]. One of the enablers of its success is the fast delivery of the orders [12], which requires an optimized, well planned and operated supply chain as well as the fast and responsive incorporation of technological

innovations. In 2017, Amazon introduced Key with in-home delivery, in order to make deliveries to the homes or garages of the customers if they are not at home [13]. Majority of the customers, however, did not like the idea of someone else entering their home, as 70% opted out of the in-home delivery only a few months later [14].

As a more preferable alternative to the in-home delivery, Amazon has introduced in-car delivery for Key in 2018, in which the delivery person delivers the order inside the car's trunk [15, 16]. In-car delivery, also referred as trunk delivery because of its delivery location in the car, was already established by DHL to make deliveries of Amazon back in 2015, but it was specific to only one car manufacturer, Audi [17]. DHL improved their in-car delivery application to include VW Polo cars 2 years later [4].

Applications of DHL and Amazon differ in terms of operating hours and information requested from the customer. DHL operates between 10am and 9pm [4] while Amazon operates between 6am and 10pm [18]. The customers can move throughout the day using their cars, but the car has to be parked in a publicly accessible place for delivery to take place. The location of the car is provided with the car's GPS, so it has to be parked in an open space in order to send the precise location. DHL customers can specify a two hour preferred delivery time window while Amazon customers may receive the order as long as their cars are available.

The emergence of the trunk delivery in the literature, however, was not until two years after its first application in real life [5]. Even after years, as of now, there are only a few works in the literature about the trunk delivery. There is also a lack of real-life based data due to the privacy concerns.

2.2 Vehicle Routing Problem with Roaming Delivery Locations

The problem introduced by enabling trunk delivery option is known as vehicle routing problem with roaming delivery locations (VRP-RDL) in the literature [19]. It is the problem of finding routes of a fleet of vehicles emerging from a central depot to the customers as they move throughout the day. Itineraries of the customers are assumed to be known. The fleet is homogeneous and its size is not limited. Each order has a size and trucks are capacitated. Unlike a city where the distance between two locations usually depends on the direction of the travel, due to the divided two-way roads, the distance matrix is assumed to be symmetric in the literature. Each customer's order is unique and can not be satisfied using another customer's order. Considering the fleet constraints and the structure of the distance matrix, VRP-RDL is a generalized, symmetric and capacitated VRP with time windows with homogeneous fleet and single depot [20, 21, 22, 23, 24, 25, 26]. Furthermore, we assume that split deliveries are not allowed.

In the following sections, the characteristics of VRP-RDL that relates it to a Generalized VRP and a VRP with time windows are explained in detail.

2.2.1 Relation with Generalized VRP

Generalized VRP (GVRP) is a variant of VRP [20]. In VRP, each customer has exactly one node in which the visit should take place, however, customers in GVRP may have more than one node and visiting any one of them is enough. This results in each customer forming a cluster of nodes as shown in Figure 2.1, an example solution is shown in Figure 2.2.

Because the deliveries can only take place if the customer's car is parked, the parking locations throughout a customer's itinerary can be replicated as the delivery points for the same customer but on different locations. Since it is enough

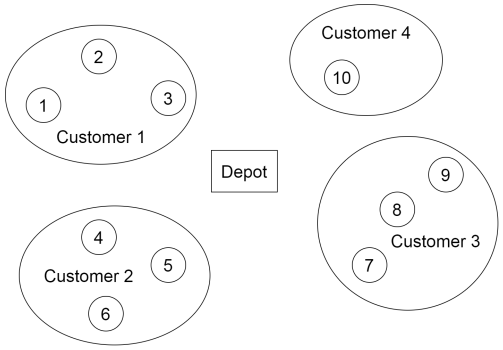

Figure 2.1: Example for a Generalized VRP setup

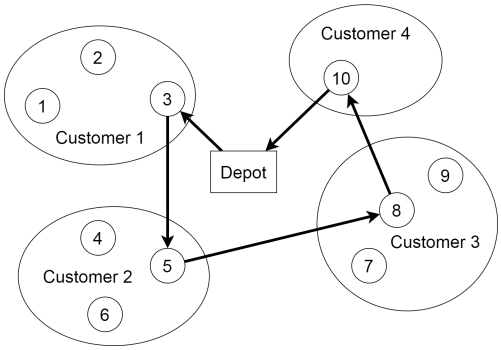

Figure 2.2: Example solution for Generalized VRP

to visit a customer's car only once, visiting any one of these replicated delivery points is sufficient to satisfy its customer. Thus, in order to satisfy all customers, a solution for VRP-RDL requires visiting one of the nodes inside each cluster formed by the customers, making it a time-windowed variant of GVRP.

2.2.2 Relation with VRP with Time Windows

In VRP with time windows (VRP-TW), each demand node has a time window during which the visit can be made to the customer and it is not possible to make visits outside of the time windows [24]. Moreover, in addition to the time-window information, VRP-TW requires the time required to traverse each arc between the nodes. It can be calculated using the location coordinates, customer movement speed and the truck movement speed. The connection between VRP-TW and VRP-RDL is established by the itineraries of the customers as delivery trucks need to 'catch' them while their cars are parked, which results each customer to form a discretely located set of nodes, each corresponding to a unique location that their car was parked at, with their associated time windows; time of the parking (enter) and the exit (leave). Deliveries outside the nodes' time windows are not possible as the customers would be parked at another location or traveling.

GVRP-TW is the closest problem to VRP-RDL, with one key difference: the former can have time windows intersecting with time windows of other nodes in

the same cluster. In VRP-RDL, however, time windows in each demand cluster inherently do not overlap, because the customer can not exist at more than one location at any given time. As an example, one may consider the GVRP-TW setup shown in Figure 2.3 and the VRP-RDL setup in Figure 2.4.

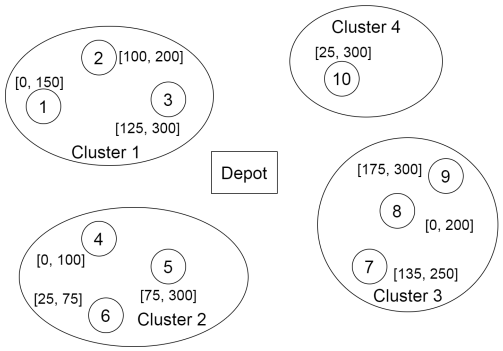

Figure 2.3: Example GVRP-TW problem

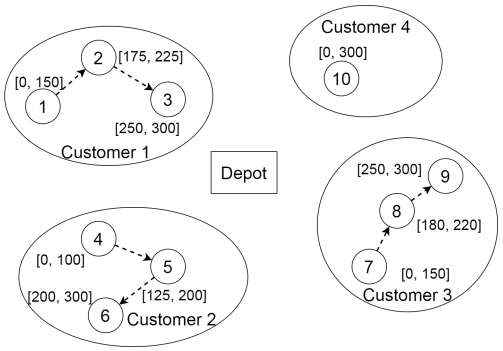

Figure 2.4: Example VRP-RDL problem with customer itinerary arcs

Formally, VRP-RDL can be summarized as finding routes that visit each customer once with respect to capacity and time-window constraints during the operational hours of the day. The challenging and beneficial part of the problem compared to the conventional capacitated VRP comes from the fact that customers are not stationary and are moving throughout the day, having small time intervals at different locations, during which the deliveries can be made. The time window and multiple available delivery locations per customer make VRP-RDL a special case of GVRP-TW where each customer location set has distinct time windows that do not intersect with each other.

Chapter 3

Literature Review

Due to its recent emergence, there are only a few studies about VRP-RDL/GVRP-TW in the literature. This chapter provides an overview of existing solution methodologies in the literature for them as well as the applications of the proposed solution methodology in the literature to the similar problems.

3.1 VRP with Roaming Delivery Locations

There are only a few exact and heuristic solution methodologies in literature for VRP-RDL. It is NP-hard as it is a special case of VRP [27].

GVRP-TW was first examined by Moccia et al.[28], even though it does not impose special conditions on time windows as it is in the VRP-RDL, the proposed incremental tabu search algorithm can still be applied to the instances of VRP-RDL.

VRP-RDL was first examined by Reyes et al.[5]; they formulated a mathematical model and a dynamic program that uses a variable neighborhood search algorithm, they also created a data-set for the problem. The underlying neighborhood structure in their proposed algorithm is similar to the one that was

proposed by Moccia et al.[28], as both consider the removal of each customer and insertion to anywhere on each route. Furthermore, they have shown that route of a fixed sequence of customers can be solved to optimality through a dynamic program.

Özbaygın et al.[19] provided an exact solution methodology, a branch and price algorithm. Their approach was able to solve instances with up to 60 customers to optimality. For instances with 60 customers, it sometimes failed to find a solution due to the time limit and it failed in all of the instances with 120 customers [19], due to the computer related constraints.

Dumez et al.[29] introduced a multi-objective version which considers the customer preference over the delivery locations and enables deliveries made to the shared delivery locations such as parcel lockers, with a capacity. Objective is to minimize the number of delivery trucks used as well as the total distance traveled in a lexicographic order. A minimum ratio for each preference level, which indicates the overall number of deliveries to be made on the locations of corresponding preference level, is provided by the decision maker. It also introduces the time required to make deliveries as a separate variable. Variable neighborhood search with a different set of ruin/recreate operators is proposed as the solution methodology.

Both the variable neighborhood search and branch and price algorithms are, however, heavily affected by the size of the problem in terms of customers and delivery points (nodes). Specifically, the solution construction phase of the variable neighborhood search algorithm may require an enormous amount of time as it grows both with the number of customers, number of nodes and the minimum number of trucks required, which may grow with the other two again. In real life, the problem may include millions of different orders, placed by thousands of customers [30], which may reduce the applicability of these solution methodologies significantly, requiring customers to be grouped in order to obtain a solution within the operating hours. This causes two significant issues; a methodology needs to be developed to group customers and the performance is inherently reduced due to the inefficiency of the grouping.

3.1.1 Stochastic Perspective

In this section, works in the literature that examine a version of VRP-RDL with a stochastic parameter is provided. In the current literature, there are only two variants of VRP-RDL with a stochastic parameter: stochastic travel times and uncertain customer movements. The version with uncertain customer movements is also referred as the dynamic version of the problem as the actual customer movements are not known from the beginning of the day but instead are provided during the day as they happen [31].

There are two papers examining a version of VRP-RDL with stochastic travel times: Lombard et al.[32] proposed a Monte-Carlo method and an enhanced greedy adaptive search procedure, Sampaio et al.[33] proposed a two stage stochastic model. These papers, however, only consider the stochasticity caused by the traffic, not by the movements of the customers themselves, i.e. locations visited by the customers and time-windows are deterministic and known from the beginning. Moreover, because the time-windows are assumed to be deterministic, the stochastic travel time matrix does not affect customers while traveling through their itineraries, preventing them from arriving early or late to their destination.

Özbaygın et al.[31] is the only one to address the problem under uncertain customer movements. They developed a methodology based on a branch and bound algorithm. The uncertainty in the customer movements is addressed by assuming a prediction to make them deterministic, which can be provided by another methodology or a company. The problem is solved dynamically in the case of an itinerary information update. It is worth noting that this version of the problem can address stochastic travel times as well, by assuming a prediction on them and updating the solution dynamically.

This thesis makes three contributions compared to the existing literature. First, it introduces a time-effective heuristic method to represent and evaluate the solutions, which can also be applied to the GVRP-TW without the special

condition on time windows. Second, a solution improvement framework independent of the problem size is introduced. Third, an algorithm to fine-tune the parameters of the framework for a provided set of problems is introduced.

3.2 Genetic Algorithm

In this section, an overview of the literature on the genetic algorithm (GA), its definition, advantages and disadvantages over the other approaches are provided. To our knowledge, there are not any work involving a GA and VRP-RDL or GVRP-TW in the literature. There are, however, various applications of GA to VRP-TW and GVRP, which are described in this section.

3.2.1 Description

First introduced by Holland[34] and inspired by the evolutionary processes observed in the nature, GAs are meta-heuristics that are able to construct and improve solutions. It is one of the first proposed algorithms that involve a population-based approach and a stochastic nature. It scans the solution space by maintaining a pool of diverse solutions, also called chromosomes. Chromosomes are usually a vector or matrix containing numbers that can be translated into a solution using a method, called the fitness function. The chromosomes usually work with the most important decision variable(s), such as the sequence of visited nodes in a VRP, rather than the other decision variables that are determined by others such as capacity of vehicle visiting a route. The remaining decision variables that can be determined by the ones that are obtained from the chromosome are calculated via a fitness function, which constructs a feasible solution following a procedure.

3.2.2 Advantages

3.2.2.1 Advantages of Utilizing Chromosomes as Solutions

Transformation of decision variables to chromosomes enable a faster solution altering process, with a reduced Hamming/Levenshtein distance, which are defined as the number of inversions and swaps required to transform solutions [35, 36]. As an example, consider a traveling salesman problem [37] with a solution represented by a 2-dimensional binary matrix indicating which arcs between the nodes should be traversed, and a vector indicating in which order should the nodes be visited. In order to swap two nodes in the sequence, matrix representation requires 8 inversion operations, or 4 swap operations, which results in a Hamming distance of 8, or a Levenshtein distance of 4, while the vector representation can perform this in a single swap operation, or two updates, resulting in a Levenshtein distance of 1 and a Hamming distance of 2, as shown in Figures 3.1, 3.2 and 3.3.

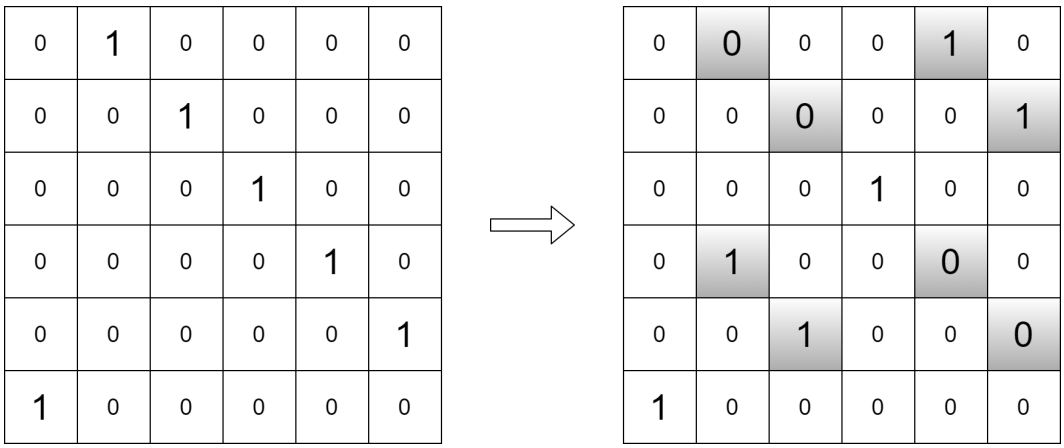

Figure 3.1: Elements of binary matrix needed to be inverted to perform a swap of two nodes

In addition to the reduced number of calculations to be done, the vector representation also does not require pre-processing to determine which operations should be performed: matrix representation needs to find out the neighbors of the nodes to determine which arcs have to be changed. A lower Hamming and Levenshtein distance not only enables a faster solution modification but also enables algorithms to search the solution space easier, as it inherently reduces the

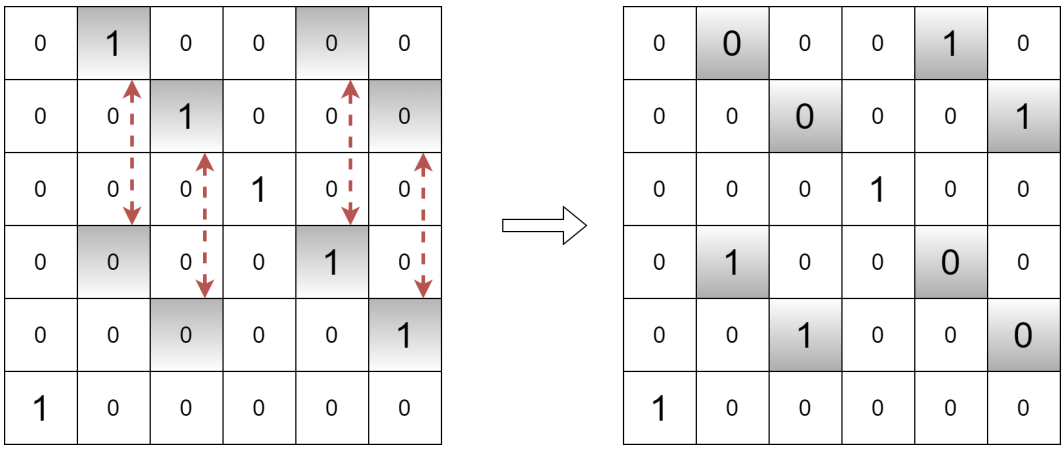

Figure 3.2: Elements of binary matrix needed to be swapped to perform a swap of two nodes

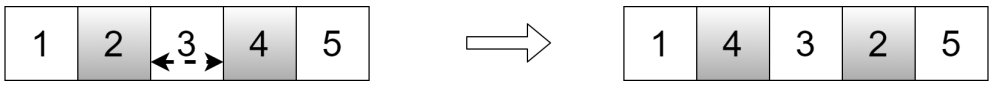

Figure 3.3: Elements of vector representation needed to be swapped/changed to perform a swap of two nodes

pre-processing overhead. This example, however, is a special case and may not be generalized to every problem, TSP was chosen due to its simplicity and relation to the VRP-RDL.

3.2.2.2 Advantages of Genetic Algorithm Structure

The algorithm avoids local optima by allowing poor-performing solutions, which may be closer to the better-performing solutions, in the pool [38]. GA relies on an iterative search process and can be allowed to run indefinitely, which also classifies it as a meta-heuristic. Throughout its progress, the focused part of the solution space changes depending on the evaluated solutions, towards the better-performing parts of the solution space. The operator-based nature of the GAs makes it easier to be combined with other algorithms such as tabu search, particle swarm optimization and simulated annealing [39, 40, 41].

The robust and flexible nature of the GA makes it applicable to many different problems, from image processing to job shop scheduling and training neural

networks [42, 43, 44]. The population of solutions in GA can be evaluated via parallel computing, which enables a much more efficient processing and execution of the algorithm, this, in turn, enables GA to be one of the most successful meta-heuristics for solving the vehicle routing problems [27].

3.2.3 Comparison with other heuristics

There are many comparative studies involving GAs but only three of them involve a problem close to the operations research literature, while the rest of the studies consider cryptography focused problems. A study found that GA is able to find better solutions compared to the simulated annealing algorithm for solving the traveling salesman problem [45]. Another study concluded that tabu search performed better than GA and simulated annealing in a microsimulation modeling [46]. The last study considered a quadratic function and concluded that GA was superior to the tabu search and simulated annealing [47].

As discussed in Section 3.2.2, one of the most significant benefits that GAs can bring is the utilization of parallel computing while evaluating the quality of the solutions inside the population. None of these comparative studies, however, utilized a parallel computing scheme. Moreover, there are two important factors that may affect a GAs performance significantly: its code structure and the way algorithm operators are implemented. In terms of code structure, since almost every operation is repeated for each solution inside the population, computations independent of the solution should be avoided inside those operations. Since almost every operator involves a stochastic action, the mathematical structure of these stochastic variables can be analyzed to perform them faster in a computer environment, such as sampling from Binomial(n, p) once instead of sampling n Bernoulli(p) distributions.

3.2.4 Applications for VRP-TW and GVRP

There are a number of studies involving a GA application to the VRP-TW, many of them propose a hybrid approach by combining it with other heuristics. In Table 3.1, an up-to-date list of literature involving GA and VRP-TW is provided. The demand column indicates whether the demands of the delivery nodes are known from the beginning (deterministic) or they are not known prior to the operational stage (stochastic). Objective and objective type columns indicate the objective of the problem, many of the works include a single objective consisting of distance and penalty costs while a few of them take the number of trucks used into account as a separate objective. Modified parts column indicate the genetic algorithm parts that are unique in the literature for each study. Finally, the truck capacity column indicates whether the delivery trucks of the problem have a capacity, most of the works consider a system with homogeneous trucks and capacity while a few of them consider trucks with different capacities and trucks with no capacities at all.

GVRP was first introduced by Ghiani et al.[68], there are only a few applications involving GA as the solution methodology in the literature. Pop et al.[69] proposed a GA and Pop et al. [70] developed a hybrid GA with local search procedures in another study.

Table 3.1: Literature review on VRP-TW with GA

Study	Dem.	Travel Times	Obj.	Obj. Type	Mod. Parts	Truck Cap.
Mak et al.[48]	Sto.	Det.	T.D. + L.P.	S.O.	✗	✓hom.
Pierre et al.[49]	Det.	Det.	T.D. and N.T.	M.O.	✓c.o.	✓hom.
Berger et al.[50]	Det.	Det.	N.T. then T.D.	M.O. Lex.	✓pop.	✓hom.
Tan et al.[51]	Det.	Det.	T.D.	S.O.	✓mut. and c.o.	✓hom.
Zhu et al.[52]	Det.	Det.	N.T. then T.D.	M.O. Lex.	✓pop.	✓hom.
Jung et al.[53]	Det.	Det.	T.D.	S.O.	✓c.o.	✓hom.
Ombuki et al.[54]	Det.	Det.	T.D. and N.T.	M.O.	✓pop.	✓hom.
Jung et al.[55]	Det.	Det.	T.D.+F.T.C.+L.P.	S.O.	✗	✓het.
Nazif et al.[56]	Det.	Det.	T.D.	S.O.	✓c.o.	✓hom.
Chang et al.[57]	Det.	Det.	T.D.	S.O.	✓mut. and c.o.	✗
Xu et al.[58]	Det.	Det.	T.D.	S.O.	✓c.o.	✓hom.
Xu et al.[59]	Sto.	Det.	T.D. + L.P.	S.O.	✗	✓hom.
Berger et al.[60]	Det.	Det.	N.T. then T.D.	M.O. Lex.	✓c.o.	✓hom.
Wang et al.[61]	Sto.	Det.	T.D. + L.P.	S.O.	✓c.o.	✓hom.
Ghoseiri et al.[62]	Det.	Det.	T.D. and N.T.	M.O.	✓i.p.	✓hom.
Alvarenga et al.[63]	Det.	Det.	T.D.	S.O.	✓i.p.	✓hom.
Li et al.[64]	Det.	Det.	T.D. + L.P.	S.O.	✗	✓hom.
Potvin et al.[65]	Det.	Det.	T.D.	S.O.	✓c.o.	✓hom.
Thangiah et al.[66]	Det.	Det.	T.D.	S.O.	✗	✓hom.
Vidal et al.[67]	Det.	Det.	T.D.	S.O.	✓c.o.	✓hom.

Dem.: Demand, Obj.: Objective, Mod. Parts: Modified Parts, Truck Cap.: Truck Capacity

Sto.: Stochastic, Det.: Deterministic, hom.: homogeneous, het.: heterogeneous, S.O.: Single Objective, M.O.: Multi-objective, M.O. Lex.: Lexicographic M.O., c.o.: Crossover, mut.: Mutation, pop.: Population, i.p.: Initial Population.

T.D.: Total Distance, L.P.: Lateness Penalty, N.T.: Number of Trucks used, F.T.C.: Fixed Truck Cost

Chapter 4

Mathematical Model

In the literature, there are two mathematical models for VRP-RDL: a mixed integer and a set partitioning formulation obtained by applying Dantzig-Wolfe decomposition [19]. In this thesis, the mixed integer formulation proposed by Özbaygın et al.[19] is used as the basis. The parameters and the decision variables of the problem that are referred throughout this thesis are given in Table 4.1.

Table 4.1: Modeling parameters and decision variables

Problem Parameters		
N	Set of nodes	
A	Set of arcs	
C	Set of customers	
Q	Truck capacity	
T	Day length	
(e_i, l_i)	Time window of node i	$\forall i \in N$
N^c	List of nodes visited by customer c	$\forall c \in C$
C^i	Customer of node i	$\forall i \in N$
t_{ij}	Time to traverse the arc between nodes i and j	$\forall (i,j) \in A$
d_{ij}	Cost (distance) to traverse the arc between nodes i and j	$\forall (i,j) \in A$
q_i	Demand of node i's customer	$\forall i \in N$
Decision Variables		
x_{ij}	1 if arc (i,j) is used, 0 otherwise	$\forall (i,j) \in A$
s_i	Arrival time of truck at node i	$\forall i \in N$
y_i	Used capacity of truck at node i	$\forall i \in N$

$$\min \quad \sum_{(i,j)\in A} x_{ij} d_{ij} \tag{4.1}$$

$$\sum_{j\in N} (x_{ij} - x_{ji}) = 0 \qquad \forall i \in N \tag{4.2}$$

$$\sum_{j\in N^c} \sum_{i\in N\setminus\{j\}} x_{ij} = 1 \qquad \forall c \in C \tag{4.3}$$

$$s_j \geq s_i + t_{ij} x_{ij} + (e_j - l_i)(1 - x_{ij}) \qquad \forall (i,j) \in A, j \neq 0 \tag{4.4}$$

$$s_i + t_{i0} x_{i0} \leq \min\{l_i + t_{i0}, T\} \qquad \forall i \in N \tag{4.5}$$

$$e_i \leq s_i \leq l_i \qquad \forall i \in N \tag{4.6}$$

$$y_j \geq y_i + q_i - Q(1 - x_{ij}) \qquad \forall (i,j) \in A, j \neq 0 \tag{4.7}$$

$$0 \leq y_i \leq Q \qquad \forall i \in N \tag{4.8}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{4.9}$$

In the model, node 0 represents the depot node. Equation (4.1) is the objective calculated as the sum of the distance traveled. Constraint (4.2) conserves the flow of vehicle routes. Constraint (4.3) ensures that each customer is visited at one of their nodes. Constraints (4.4)-(4.6) calculate the arrival times of the trucks at nodes, limit each one to be within the associated time-window and make sure that truck arrives at the depot before the end of the day. Constraints (4.7) and (4.8) limit and determine the capacity of the vehicles at the nodes.

4.1 Operational Dynamics and Related Assumptions

In this section we provide the assumptions made in VRP-RDL while evaluating the proposed solution methodology. It should be noted that, however, some of these assumptions are due to the problem instances available in the literature and the proposed solution methodology can still work without these assumptions. Inherent assumptions due to the available problem instances are as follows:

- Travel speed is same for every customer.
- Trucks travel at the same speed as customers.
- Travel time matrix is constant throughout the day, not taking traffic into account.

Inherent assumptions due to the proposed solution methodology are as follows:

- No split delivery is allowed.
- No additional service time while delivering the order on the node.
- No upper limit on the number of trucks that can be utilized.

There are, however, workarounds for all of the assumptions made by the proposed solution methodology, which are provided in Section 5.2.4.

Chapter 5

Solution Methodology

In this chapter, description of the proposed solution methodology, as well as a fine tuning algorithm to improve its parameters are provided. The proposed solution methodology is a genetic algorithm (GA) based meta-heuristic. In the following sections, the general nature of GAs, GA for obtaining solutions for the problem (Solver-GA: SGA) and GA for fine-tuning the SGA (Fine-tuning-GA: FGA) are explained in detail.

5.1 Genetic Algorithm

Genetic algorithms are composed of evolutionary operators working collaboratively. The operator-based structure provides a highly customizable framework that can be applied to most types of problems. In this section, the general working principle of GAs and the underlying mechanisms and operators are described in detail.

5.1.1 General Structure

Most applications of GA usually involve the steps shown in Figure 5.1. Since GAs are population based algorithms, they require a population consisting of solutions to start, called the initial population. Initial populations can be either constructed using random solutions or obtained from another algorithms including but not limited to Tabu Search, Simulated Annealing and Particle Swarm Optimization. Once the population is ready, the algorithm evaluates the quality of the solutions inside and generates the next population by selecting parent pairs, survivors and elites from the population, based on the quality of the solutions. Once the next population is ready, the same steps are followed until the terminating conditions are met.

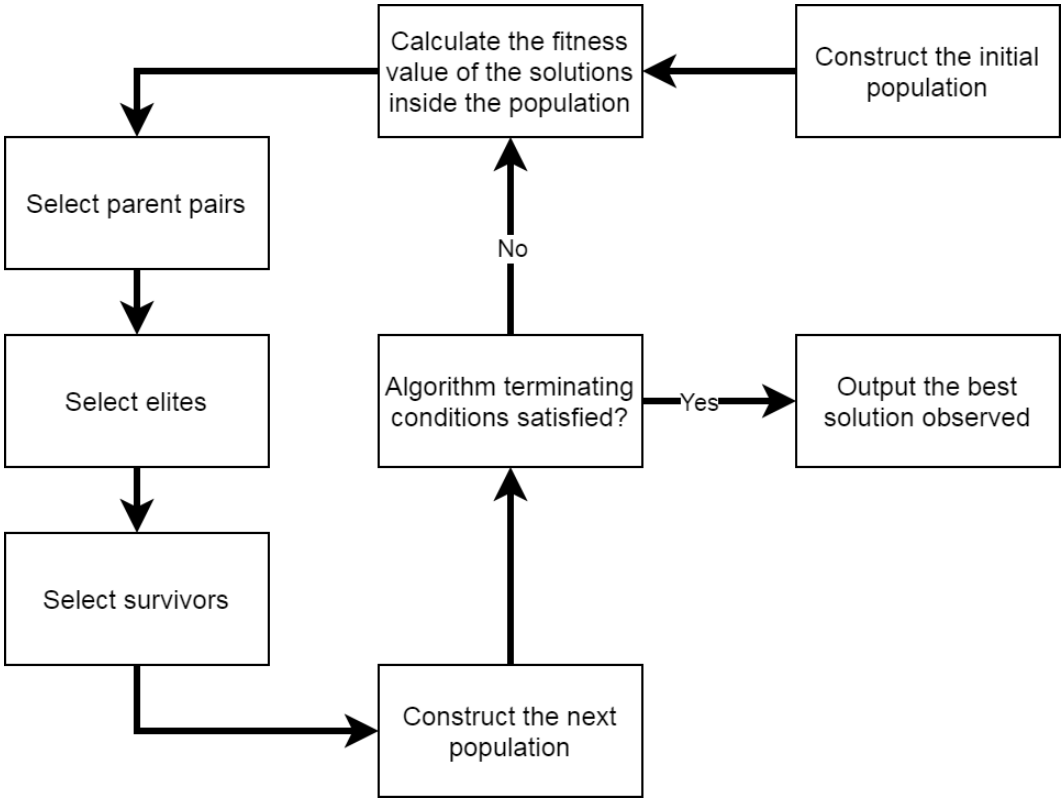

Figure 5.1: Flowchart of GA

5.1.2 Solution Representation and Fitness Function

In order to operate on solutions fast and in a structural manner, solutions are encoded in and being worked on using a representation, also known as chromosomes. A chromosome is usually a vector consisting of numbers that can be translated into a solution for the problem. It should be noted that a chromosome can consist of anything as long as the other operators are able to work with it. Primitive objects that make up the chromosome are called genes. An example overview is provided in Figure 5.2.

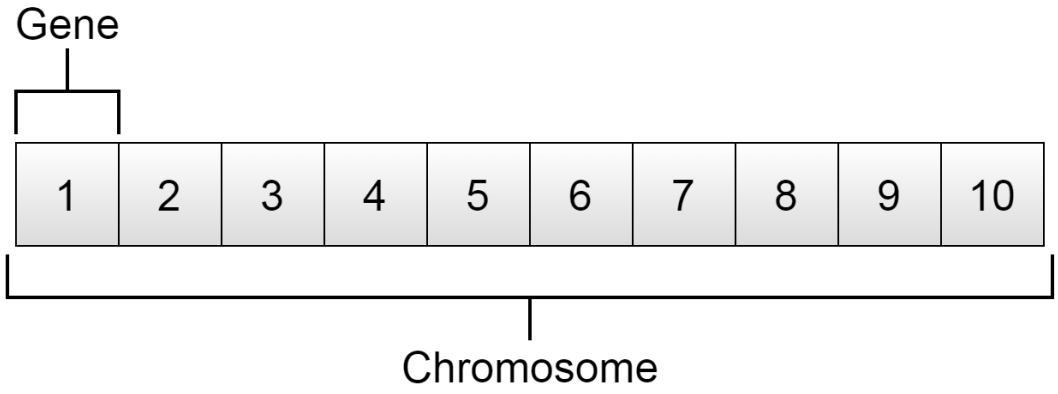

Figure 5.2: Example of chromosome and gene on an encoded solution

Depending on the nature of the problem, the representation may result in infeasible solutions, which can be dealt with either including a penalty in the objective or repairing the chromosome. In order to avoid dealing with infeasible solutions, a solution representation may employ some rules, specific to the problem, such as lower/upper bound on genes or number of genes with some value observed in the chromosome. These rules result in different types of solution representations, there are three commonly used types: continuous, where genes are real-valued and subject to lower/upper bounds, permutation, where chromosome is a permutation of a specific set of values, and binary type, which is composed of binary genes. A fitness function is utilized in order to assess the solution quality of each chromosome. The solution quality, also known as fitness value/score, usually is the objective value of the solution constructed from the chromosome, including any penalty caused by the infeasible parts.

The solution representation is one of the most significant factors that affect the algorithm performance, specifically, it affects how fast the algorithm can operate and which parts of the solution space is being searched. Depending on the problem, algorithms may spend over 50% of the processing time while decoding the chromosomes into solutions and obtaining the fitness values. Because the solutions are constructed by the fitness function utilizing the solution representation, the search space is directly determined by them.

5.1.3 Initial Population

GA is one of the population-based heuristics and require an initial population of solutions/chromosomes to begin its processes. The initial population can be filled either randomly or by using output obtained from other algorithms. Quality of the solutions inside the initial pool, as well as their distribution on the solution space is one of the factors affecting the algorithm output [71].

5.1.4 Constructing the Next Population

In order to move the search space, a new population is built using the previous population. Each population significantly affects the one that comes after, in a stochastic manner. Solutions of the current population have chances of having their characteristics (parts of their chromosome) to be transferred into the next generation directly proportional to their fitness values, compared to the rest of the population. Most of the GA applications follow a different procedure at this step of the algorithm.

The next population is usually composed of three types of individuals: offspring, survivors and elites. Independent of the type, each set of individuals are selected from the existing population using selectors. The definitions of selectors and types of individuals are provided in the following sections.

5.1.4.1 Selectors

There are various types of selectors and they can be classified considering two factors: stochastic or deterministic and score or rank based. Stochastic selectors employ a randomness at a point during the selection process while the deterministic ones involve a deterministic selection, picking a number of the best solutions. Score based ones assign a probability proportional to the fitness scores whereas the rank based ones map the fitness scores to discrete probabilities depending on their rank in the population. An overview of the most commonly used selectors is provided in Table 5.1.

Table 5.1: Commonly used selectors

Selector Name	Stochastic	Deterministic	Score Based	Rank Based
Tournament[72]	✓	✓	✗	✓
Roulette Wheel[73]	✓	✗	✓	✗
Truncation[74]	✗	✓	✗	✓
Exponential Rank[75]	✓	✗	✗	✓
Linear Rank[75]	✓	✗	✗	✓
Reward-based[76]	✓	✗	✓	✗
Boltzmann[77]	✓	✗	✓	✗

5.1.4.2 Offspring & Crossover and Mutation Operators

Offspring are constructed using a crossover operator on the selected parent pairs. A parent pair is generated by the parent selection operator, the first parent is not considered as a candidate while picking the second parent to avoid matching the same individuals with themselves. Crossover operators are specific to the types of solution representations in order to produce chromosomes that do not violate the solution representation rules, described in Section 5.1.2. There are only three crossover operators for permutation chromosome type due to its strict rules [78, 79], they are based on swapping an interval between the parents then repairing each chromosome. There are many crossover operators for the GAs

with continuous or binary chromosome types [78], including multi-point, single-point and uniform crossover [80, 81, 82]. As long as it results in two different individuals using the parts of the parents, any algorithm/procedure can be used as a crossover, enabling developments of problem-specific algorithms to enhance the overall GA performance.

The offspring obtained from the crossover operator are then passed to the mutation operator, which takes a chromosome and applies a random noise. The way that random noise is being applied to depends on the solution representation rules: permutation chromosome types usually employ a random swap operator while the other types may employ Gaussian noise, uniform noise or inversion [83, 84]. Because they are altered, new solutions, the resulting offspring have to get their fitness value calculated again.

5.1.4.3 Survivors and Elites

The selected survivors and elites are transferred directly, without undergoing any alteration process, to the next population. From computational perspective, because they are not modified, they save computational time and enable processing more generations. From an algorithmic perspective, they act as anchor solutions left in the better parts of the solution space and enable the algorithm to perform search around these regions. The main difference between the survivors and elites is that the former is selected in a stochastic manner, using one of the selectors in Table 5.1, while the latter is simply the best ranking solutions in the population. The elites are excluded from the survivor candidate pool to prevent having copies of an individual in the next population, and to preserve the variance of the population.

5.1.4.4 Algorithm Termination

Once the offspring are created and their fitness values are calculated, they are merged with the selected elites and survivors to construct the next population.

The algorithm can process new populations indefinitely, therefore, before continuing with the next population, the terminating conditions are checked. There are a number of different terminating conditions, they can be combined and are selected depending on the application, most common ones are provided in Table 5.2.

Table 5.2: Commonly used terminating conditions

Condition Name	Time-based	Population-based	Solution-based
Time limit	✓	✗	✗
Population convergence	✗	✓	✗
Generation limit	✗	✓	✗
Solution convergence	✗	✗	✓
Fitness value limit	✗	✗	✓

5.2 Solver GA (SGA)

The SGA inherits the basic GA structure discussed in Section 5.1 and implements a solution representation and fitness function specific to the problem. In addition to the core GA structure, SGA employs a local convergence prevention procedure, which randomizes a proportion of the current solutions in the population, to avoid converging to a local optima. In this section, the proposed SGA with its implementation to VRP-RDL is described in detail.

5.2.1 Solution Representation and Fitness Function

A solution is a vector composed of all node indices in the problem and stop-signals (equivalent to depot index) and is represented using a permutation representation, i.e. each value is unique throughout the vector. The sequence of values inside the vector determines in which order nodes should be visited. If a visit determined to be infeasible, the current truck is sent to the depot and a new truck is dispatched to follow the rest of the sequence. The node is simply skipped if its customer was already visited at another node. Stop-signals enable having short routes that

would otherwise be impossible using this representation, which also enable whole solution space to be covered using this representation, as any sequence of nodes can be represented.

In Figure 5.3, an example solution represented as vector and it is possible route plan are shown in a VRP setting.

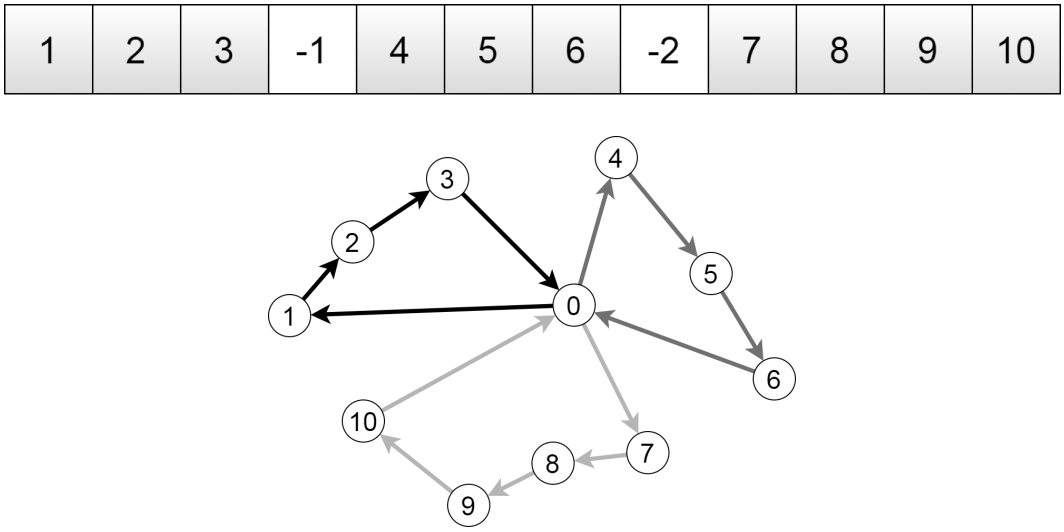

Figure 5.3: Example solution vector and resulting routes

To provide a complete perspective, a similar solution vector and the constructed routes for VRP-RDL are shown in Figure 5.4. Iterations of the algorithm can be seen from successful (normal lines) and failed routes (dashed lines). The failed routes are the visits considered by the algorithm while iterating through the vector but were found to be infeasible (constraint violation) or unnecessary (customer already visited).

The construction stage for the example in Figure 5.4 can be summarized as follows: Node 1 is visited first by arc 1, nodes 2 and 3 are skipped because their customers were visited before. Node 4 was found to be feasible for a visit, forming arc 2, nodes 5 and 6 are skipped because of the customer rule, there is a negative value after them, so the truck is routed back to depot by arc 3, finishing its route. A new truck is utilized, capacity and time-window values are reset, the node 7 is visited, nodes 8 and 9 are skipped. Node 10 was found to be infeasible to visit, so the truck at node 7 is sent back to depot and a new truck is sent to node 10,

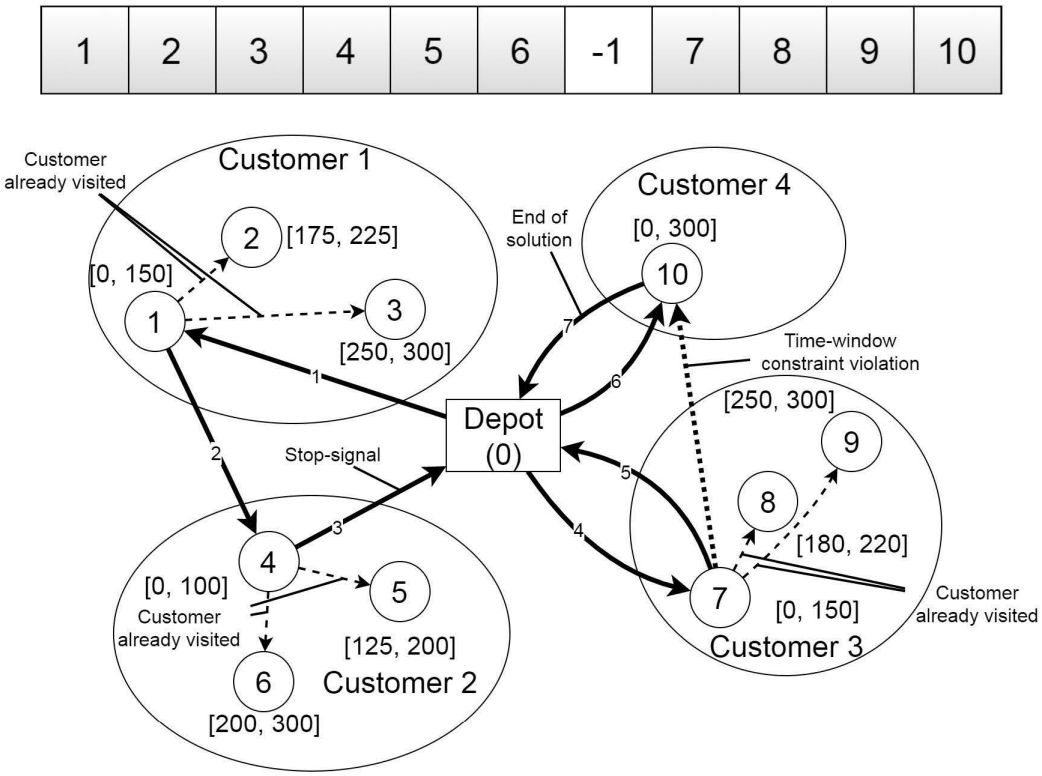

Figure 5.4: Example solution vector and constructed routes

which is at the end of the solution vector, so the truck is sent back to the depot afterwards.

The routes are constructed by reading the nodes inside the vector from left to right, sending the current truck back to the depot (ending the route) if the current node index is negative or the next node can not be visited due to time window or capacity constraints. Trucks stay at the current node in case the next node's customer was already visited before. The time window constraints consider both the customer's arrival to the node and truck's arrival to the depot after the delivery. The pseudo-code for route construction using the vector representation and the calculation of the fitness function can be found in Algorithm 1.

Algorithm 1: Fitness function to evaluate a solution

Input: Solution vector: $\mathtt{X}$, Model variables

Output: Routes: $\mathtt{R}$, Total cost (distance traveled): $\mathtt{Y}$

/* Initialization */

$\mathtt{R} = \emptyset$

$\mathtt{Y} = 0$

$\mathtt{V} = \emptyset$ // set of visited customers

/* Variables of the current truck */

$\mathtt{r} = \{0\}$ // current route, emerging from depot

$\mathtt{k} = 0$ // last visited node

$\mathtt{q}^* = 0$ // capacity

$\mathtt{t}^* = 0$ // time

for ($\mathtt{i} \in X$) {

  if ($\mathtt{i} < 0$) {

    /* stop-signal, set destination as depot */

    $\mathtt{i} = 0$

  $\mathtt{node\_customer} = C^{\mathtt{i}}$

  $\mathtt{t}' = \max(\mathtt{t}^* + \mathtt{t}_{\mathtt{ki}}, \mathtt{e}_{\mathtt{i}})$ // delivery time

  $\mathtt{t}'' = \mathtt{t}' + \mathtt{t}_{\mathtt{i0}}$ // arrival time in the depot after visit

  if (($\mathtt{i} = 0$ and $|r| > 1$) or ($\mathtt{node\_customer} \notin V$ and $\mathtt{t}' \not> \mathtt{l}_{\mathtt{i}}$ and $\mathtt{t}'' \leq \mathtt{T}$ and $\mathtt{q}^* + \mathtt{q}_{\mathtt{i}} \leq \mathtt{Q}$)) {

    $\mathtt{r} = \mathtt{r} \cup \{\mathtt{i}\}$

    $\mathtt{Y} = \mathtt{T} + \mathtt{d}_{\mathtt{ki}}$

    if ($\mathtt{i} = 0$) {

      $\mathtt{t}^* = 0$

      $\mathtt{q}^* = 0$

      $\mathtt{R} = \mathtt{R} \cup \{\mathtt{r}\}$

      $\mathtt{r} = \{0\}$

    else

      $\mathtt{t}^* = \mathtt{t}'$

      $\mathtt{q}^* = \mathtt{q}^* + \mathtt{q}_i$

    $\mathtt{k} = \mathtt{i}$

  else

    /* can not make the visit or trying to visit depot from depot, skip */

return $\mathtt{R}$, $\mathtt{Y}$

5.2.1.1 Number of Stop-signals

The stop-signals became a necessity after a preliminary analysis on the optimal and near-optimal solutions obtained using the mathematical model. In many instances, optimal solutions included a few short routes, visiting only 2-3 customers and spending most of the time waiting instead of traveling. The solution constructor without stop-signals can construct short routes only when the nodes are ordered in a way that results in either a capacity or time window violation, making it less probable. The stop-signals in the solution vector force the current truck, that would otherwise continue visiting the nodes until it can not visit the next one due to capacity or time window constraints, to end the route by traveling to depot, effectively cutting the route short. However, there should be a limit on the number of stop-signals allowed inside the chromosome, in order to avoid increasing the Levenshtein distance to the closest chromosome that is equivalent to an optimum solution. Having as few of them as possible also makes the algorithm faster since it depends on the total length of the chromosome, as shown in Algorithm 1. Consider a chromosome with more than half of its genes as stop-signals, for instance, not only it will have increased number of redundant calculations but also forming routes will become less likely. Since GAs explore the solution space provided by the solution representation in a stochastic manner, likelihood of having longer routes (after altering operations) decreases with the number of stop-signals. In the proposed SGA, number of stop-signals is determined as 25% of number of the customers, leaving four customers per route on average. This number was picked by analyzing the average number of customers per truck in the model results, provided in Appendix C, which found 3.85 customers per truck, 0.26 truck per customer approximately.

5.2.2 Initial Population

The initial population is filled by shuffling a vector containing the node set $\mathtt{N} \setminus \{0\}$ and stop-signals $\{-1, -2, ...\}$. The stop signals have to be different from each other due to the rules of the chromosome, and they are set as negative numbers

for simplicity.

5.2.3 Algorithm Setup

As for the GA setup, the parameter values in Table 5.3 were determined after running the FGA, which is detailed in Section 5.3.

Table 5.3: Solver-GA parameters

Parameter name	Parameter value
Population size	100
Offspring count	44
Survivor count	4
Elite count	52
Parent selector	Tournament with size 5
Survivor selector	Linear rank with parameter as 1.91
Mutation probability	0.023
Time limit	5 minutes

5.2.3.1 Parent Selector

The parent selector, tournament selection, randomly picks 5 solutions, independent of the fitness value, from the population as the competitors. The best solution among the competitors is then chosen as the winner, the parent. Before selecting the next parent by the same procedure, the first selected parent is excluded from the candidate pool in order to avoid matching a solution with itself, it is again included in the candidate pool of the next parent pair.

5.2.3.2 Survivor Selector

The survivor selector, linear rank selection, is a rank based selection algorithm with one parameter, p, which is proposed as 1.91. Weight of the best solution is p while the worst solution has a weight of $2-p$, weights of the rest of the solutions are distributed evenly, in the order of their ranks, between p and $2-p$. Therefore,

a solution with rank r (where best solution has $r = 1$) will have a weight of $\frac{p\times(\text{population size}+1-r)+(2-p)\times(r-1)}{\text{population size}}$, which is calculated as shown in Equation 5.1 The selection probabilities are then obtained by dividing each weight by the sum of them.

$$\text{Selection weight} = \frac{1.91 \times (101 - r) + (0.09) \times (r - 1)}{100} \tag{5.1}$$

5.2.3.3 Crossover

The partially matched crossover is picked as the crossover operator due to its wide utilization in the applications with permutation chromosome types. It consists of two steps, swapping a random interval and then repairing each offspring. This operator requires each value to be unique in order to perform a consistent interval swap, so each stop-signal is included as a unique element. The two stages can be seen in Figure 5.5.

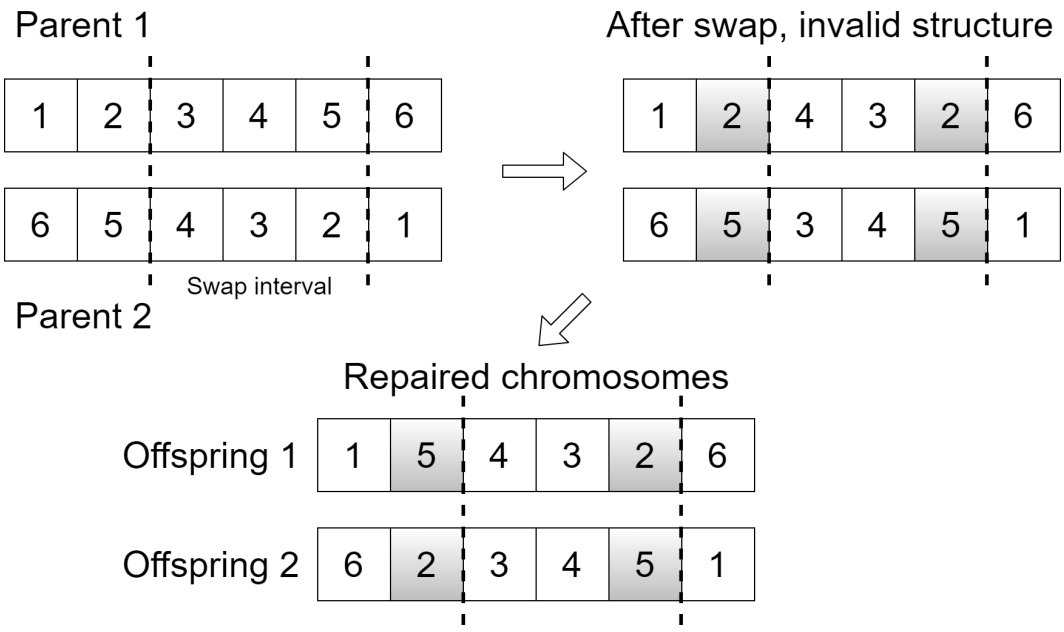

Figure 5.5: Partially matched crossover to produce the offspring from a parent pair

At the swapping step, the interval is directly swapped, which usually results

in some elements to be missing and some to be present twice. Since each element is required to exist exactly once, a repair outside the interval is necessary to fix these variables. It is done by finding the elements that exist both inside and outside the interval and selecting the ones outside. The uniqueness rule on the elements ensure that there will be same number of selected elements in both of the chromosomes. The repair is finalized by swapping the selected elements between the chromosomes, they are matched with each other in the same order they are found in the chromosomes.

5.2.3.4 Mutation

The mutation probability is proposed as 0.023, which is the probability of performing one swap operation for each gene in the chromosome. The binomial distribution was utilized in order to avoid iterating through the chromosome and save time. Pseudo-code for the proposed mutation operator is provided in Algorithm 2.

Algorithm 2: Mutation operator

Input: Solution vector: $\mathtt{X}$, Mutation probability: $\mathtt{p}$
Output: Mutated solution vector: $\mathtt{X}'$

```
r = Binomial(n = |X|, p = p)          // number of swaps to perform
X' = X
/* Perform the random swaps in chromosome                          */
for ( i ∈ {1, ..., r} ) {
    Random_Swap(X')
return X'
```

5.2.3.5 Local Convergence Prevention Procedure

Since the algorithm considers the solutions around its current population, the solutions it may find in its next population highly depends on the current population. Finding a better solution becomes especially less probable when the population is converged to or scattered around a local optimum that has a significantly large Levenshtein distance to other local and global optimal solutions.

Such situations can be prevented by checking either the diversity of the solutions in the population or the objective values of the best solutions throughout the generations. The first option requires a metric to be calculated using the population, which can be quite expensive in terms of processing time as it will depend on the length of the solutions, i.e. the problem size. The second option, on the other hand, is not affected by the problem size as it works with the objective value of the best solution. Once the population is deemed to be converged, a chosen subset of the solutions inside it can be randomized to provide algorithm with a new population, away from the previously found local optima, preferably.

The proposed local optima convergence prevention procedure utilizes the option of working with the objective values, it replaces the 95 worst solutions in the current population (95% of the population) with randomly generated ones. The best 5 solutions are left in the population to avoid losing the known valuable regions. The first randomization is done after 1.200 generations without any improvement on the best solution, and is repeated every 1.000 generations after the first randomization. The number of generations waited before the first randomization is picked higher than the number of generations in between to make sure that the first population is converged and is not able to find better solutions. These numbers were picked after a preliminary analysis of the number of generations between the solution improvements, which usually was not higher than 1.000. The procedure is detailed in Algorithm 3, which is called after the fitness function calculations are done and the objective value conditions are met.

Algorithm 3: The implemented local optima convergence prevention procedure

Input: Population: Π, Fitness values: $\mathtt{G}$
Output: Population: Π, Fitness values: $\mathtt{G}$

```
/* Select the worst 95%                                          */
worst_solution_indices = indices_of_worst(G, 0.95)
for ( i ∈ worst_solution_indices ) {
    Π_i = generate_random_solution()
    G_i = fitness_function(Π_i)                    // Algorithm 1
return Π, G
```

5.2.4 Relaxing the Assumptions

In this section we provide the workarounds that can be applied to the data and the fitness function (described in Algorithm 1) in order to implement the constraints that were not considered. The assumptions were as follows:

- No split delivery is allowed.
- No additional service time while delivering an order at a node.
- No upper limit on the number of trucks that can be utilized.

5.2.4.1 Split Deliveries

Split delivery is the practice of splitting the orders of the same customer into different delivery trucks and having each truck visit the customer in order to complete the delivery. One major benefit of the split deliveries is the increased utilization of the delivery trucks, as an example, the orders in Figure 5.6 require a total of 5 trucks to be delivered, while they can be delivered by 3 trucks using the split-setup in Figure 5.7.

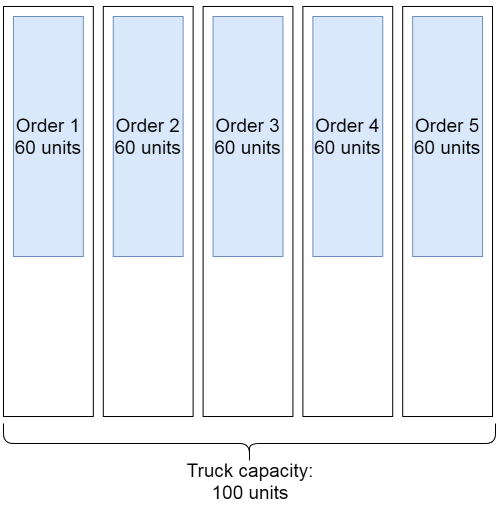

Figure 5.6: Order assignment with no split deliveries

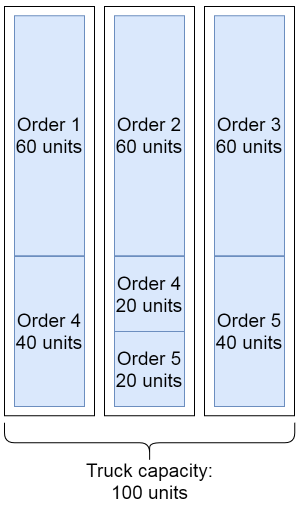

Figure 5.7: Order assignment with split deliveries

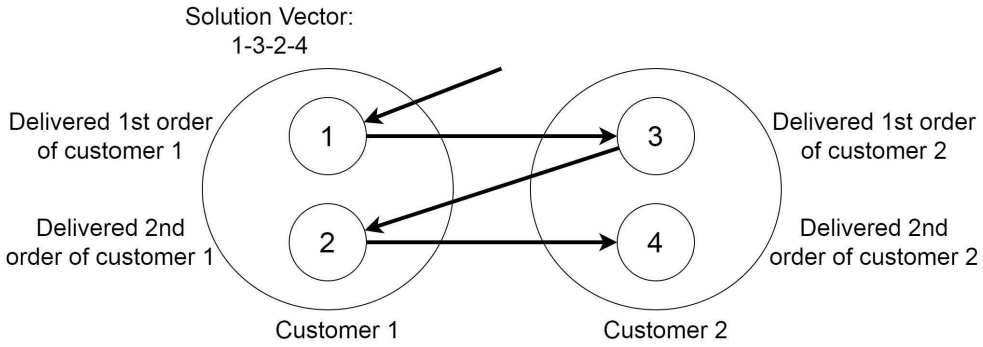

Figure 5.8: Example of an inefficient route with duplicated customers

In problems with costs or constraints related to the truck utilization, this benefit becomes especially significant. In real-life applications of e-commerce, however, split deliveries are not usually in the practice, instead, different orders by the same customer are usually consolidated and shipped as a single parcel.

In order to implement the split delivery option, a copy of each customer can be created for each sub-order in their order, where sub-orders are the parts of the order that can be delivered separately. Each customer copy will have the same itinerary as the original customer and their demand as the corresponding sub-order. This replication of the customers, however, will result in a longer solution vector as the nodes visited by the customers are replicated as well. Duplicates of the customers inside the vector may result in a truck route visiting the same customer more than once at different times, as shown in Figure 5.8, which, in turn, makes it harder for the algorithm to improve solutions due to the increased Levenshtein distance as discussed in Section 3.2.2. This leaves two options to improve the algorithm output: Allocating more time to the algorithm or post processing each constructed route in Algorithm 1 as shown in Algorithm 4.

Since the constructed routes are already checked in terms of time window and capacity constraints, they do not need to be checked for feasibility again in Algorithm 4.

Algorithm 4: Elimination of inefficient routes in an existing solution

Input: Routes: $\mathtt{R}$
Output: Routes: $\overline{\mathtt{R}}$

$\overline{\mathtt{R}} = \emptyset$
for ($\mathtt{r} \in \mathtt{R}$) {
    visited_customers $= \emptyset$
    $\overline{\mathtt{r}} = \emptyset$
    for ($\mathtt{i} \in \mathtt{r}$) {
        if ($C^{\mathtt{i}} \in$ visited_customers and $\mathtt{i} \neq 0$) {
            /* Customer was already visited before in another node; no need to visit it again in this node */
        else
            visited_customers $=$ visited_customers $\cup \{C^{\mathtt{i}}\}$
            $\overline{\mathtt{r}} = \overline{\mathtt{r}} \cup \{\mathtt{i}\}$
    $\overline{\mathtt{R}} = \overline{\mathtt{R}} \cup \overline{\mathtt{r}}$
return $\overline{\mathtt{R}}$

5.2.4.2 Service Time

The service time is the amount of time that each order requires to be delivered to customer once the delivery truck arrives at the node. Service time of node i is assumed to be γ_i throughout this section, with service time in depot node as 0 ($\gamma_0 = 0$). In case where delivery can be made as long as truck arrives between (e_i, l_i) for node i, a simple change in travel time matrix by adding γ_i to time of each outgoing arc as shown in Equation (5.2) is enough to include the service time.

$$\text{New travel time matrix} := t' = \left\{t'_{i,j} | t'_{i,j} = t_{i,j} + \gamma_i \forall (i,j) \in A\right\} \tag{5.2}$$

In case when a delivery is not feasible if the remaining time of the node is less than γ_i by the time truck reaches it, in addition to the travel time matrix modification, changing the time windows as shown in Equation (5.3) is necessary. This case, however, is highly unlikely since the customers would not leave after the delivery process is initiated. So it can be safely assumed that customer will wait for as long as the delivery truck arrives before they leave instead.

$$\text{New leave times} := l' = \{l'_i | l'_i = l_i - \gamma_i \forall i \in N\} \tag{5.3}$$

5.2.4.3 Limited Number of Delivery Trucks

The only constraint that requires a modification in the algorithm is a limit on the number of delivery trucks available, which is denoted by ν. Because the algorithm is free to utilize as many trucks as it needs, one way to include such a limit is to penalize the objective depending on the severity of the violation. Consider two solutions, one using $\nu + 1$ trucks and another using $\nu + 100$, from the problem's perspective, both of them would be considered as infeasible. From the GAs point of view, allowing such solutions by adding a fixed cost per additionally used truck would enable algorithm to distinguish solutions with less trucks and increase their selection probabilities. So it may eventually find a feasible plan but it is not guaranteed. The fixed cost, or penalty, per additional truck (ψ) should be big enough to be prioritized by the algorithm considering the distance related costs, one problem-independent way of determining it is to sum up all distance costs as shown in Equation (5.4)

$$\psi = \sum_{(i,j)\in A} d_{i,j} \tag{5.4}$$

Once ψ is calculated, it can be implemented to the fitness function in Algorithm 1 as a post-process shown in (5.5) where `R` and `Y` is the output from the fitness function and `Y'` is the penalized total cost.

$$Y' = Y + \psi \times \max\{0, |R| - \nu\} \tag{5.5}$$

5.3 Fine-tuning Algorithm

The fine-tuning genetic algorithm implements the general GA structure described in Section 5.1, with custom crossover and mutation operators specific for the fine-tuning purposes. The general structure of FGA is provided in this section, parameters of the FGA can be found in Table 5.4.

Table 5.4: Parameters of Fine-tuning GA

Parameter name	Parameter value
Population size	50
Offspring count	30
Survivor count	5
Elite count	15
Parent selector	Roulette Wheel Selection [73]
Survivor selector	Tournament Selection with size 3 [72]

5.3.1 Chromosome Structure and Initial Population

Since it is developed to derive a GA setting, each chromosome is composed of genes that are actual genetic algorithm parameters and represent a complete algorithm setting together. Therefore each gene holds an information regarding an algorithm parameter, list of genes in the chromosome and the procedure for randomly generating them are as follows:

- **Population Size:** 100.
- **Offspring, survivor and elite count:** Population size is randomly distributed to each parameter using Algorithm 5.
- **Mutation:** Uniform between 0 and 1.
- **Crossover operator:** Randomly selected as one of the following:
  - Partially Matched Crossover[78].
  - None, in order to determine whether the computational time can be utilized better without it.
- **Parent and survivor selectors:** Randomly selected as one of the following:
  - Roulette Wheel Selection[73].
  - Tournament Selection[72]. Tournament size = $\lfloor \text{population\_size} \times ratio \rfloor$, where $ratio = \text{Uniform}(0, 1)$.

– Linear Rank Selection[75]. Parameter: Uniform$(0, 2)$.

– Exponential Rank Selection[75]. Smoothing coefficient: Uniform$(0, 1)$.

– Truncation Selection[74].

The population size is kept constant in order to focus on the ratio of the offspring, survivor and elite counts. It is picked as 100 after a preliminary analysis. Since it defines the size of the search space, it should be increased with the problem size to avoid convergence to local optimum solutions, while keeping the individual type ratios the same.

Algorithm 5: Generating uniformly distributed population properties using a fixed population size

Input: population_size
Output: offspring_count, survivor_count, elite_count
/* random weight generation */
1 weights = {0, U(0,1), U(0,1), 1}
2 sorted_weights = sort(weights)
/* generating the uniformly distributed values, summing up to population_size */
3 offspring_count = $\lfloor$(sorted_weights$_1$ − sorted_weights$_0$) × population_size$\rfloor$
4 survivor_count = $\lfloor$(sorted_weights$_2$ − sorted_weights$_1$) × population_size$\rfloor$
5 elite_count = population_size - offspring_count - survivor_count
6 **return** offspring_count, survivor_count, elite_count

5.3.1.1 Fitness Function

In order to assess the performance of each algorithm setup, they are run on a set of instances and the average gap between the algorithm and the model output is considered as the fitness value. The GA outputs are, however, random due to the stochastic operators involved in the process. It is important to take the variability of the output into consideration while evaluating the algorithm settings. Therefore, each setting-problem pair are sampled 30 times and the performance of the algorithm setting over these samples are considered. The gap values are obtained for each sample and instance, the fitness value is calculated

using Equation (5.6), which computes the sum of average gaps throughout all samples, and average standard deviation per instance.

I : Selected instances $= \{1, \ldots, 30\}$

J : Sample indices $= \{1, \ldots, 30\}$

$g_{i,j}$: Optimality gap value of instance i, jth sample $\forall i \in I, j \in J$

$\overline{g}_i$: Average optimality gap value for instance i $\forall i \in I$

$$\text{Fitness value} = \frac{\sum_{i \in I} \overline{g}_i}{|I| \times |J|} + \frac{\sum_{i \in I} \sqrt{\frac{\sum_{j \in J} (\overline{g}_i - g_{i,j})^2}{|J|}}}{|I|} \tag{5.6}$$

There are 30 problem instances that were used as the basis in FGA, problem parameter information for which can be found in Appendix A. Instance selection was made based on the optimality gap values between the initially developed GA and model results, instances with the largest gap were picked. The benchmark instance problems are the same problems used by Özbaygın et al.[19], they are slightly modified versions of the instances generated by Reyes et al.[5].

Additionally, the instance-information tables include the number of nodes that are feasible to visit during the day, because of the time window constraints, a node is considered as infeasible if any of the Equations (5.7) and (5.8) does not hold.

$$t_{0i} \leq l_i \quad \text{Can make it to the node before customer leaves} \tag{5.7}$$

$$\max\{e_i, t_{0i}\} + t_{i0} \leq T \quad \text{Can make it to the depot back in time after visit} \tag{5.8}$$

5.3.2 Constructing the Next Population

The next generation is constructed by selecting elites, survivors and creating the offspring.

5.3.2.1 Survivors and Elites

Elites are selected as the best 15 solutions in the current population while the survivors are selected using the tournament selection described in Section 5.2.3.1. The tournament size is chosen as 3 and there are 5 survivors selected.

5.3.2.2 Offspring & Crossover and Mutation Operators

The same offspring generation procedure as the one described in Section 5.1 is used. Roulette wheel selection, also known as fitness proportionate selection, is picked as the parent pair selector. It is one of the selection operators that generate the selection probability depending on the fitness values, higher fitness values indicate higher chances of being selected. The fitness function described in Section 5.3.1.1, however, indicates the average deviation of the algorithm from the optimal/near-optimal solutions as well as its average output variability, so it needs to be minimized. Therefore, since roulette wheel selection works promotes the solutions with higher fitness values, the fitness values obtained by the fitness function are transformed as shown in Equation (5.9).

f_k : Fitness value of setting k, obtained by
the fitness function in Section 5.3.1.1 $\forall k \in \{1, .., 50\}$

$\overline{f}_k$: Transformed fitness value of setting k that should
be maximized to improve algorithm performance $\forall k \in \{1, .., 50\}$

$$\overline{f}_k = f_k - \max_{k \in \{1,..,50\}} f_k + \min_{k \in \{1,..,50\}} f_k \tag{5.9}$$

The transformation is applied to each population separately and, once it is applied, the selection probabilities of the individuals are defined as the proportion of their fitness value to the total fitness value of the population, which is determined in Equation (5.10).

β_k : Probability of solution k to be selected.

$$\beta_k = \frac{\overline{f}_k}{\sum_{k \in \{1,..,50\}} \overline{f}_k} \tag{5.10}$$

Once the discrete selection probabilities are calculated, one of the solutions are picked randomly as the first parent. Similarly to the SGA, the first selected parent is removed from the candidate pool before selecting the second parent, in order to avoid matching a solution with itself. The selection probability values of the population are calculated again and then the second parent is selected the same way as the first parent.

The crossover operator consists of different operators for each algorithm parameter, in order to perform a meaningful recombination. In order to avoid extreme differences between the parents and the offspring, crossover on each parameter is performed with an assigned, specific probability. The detailed explanation for the crossover on each algorithm operator and parameter is provided below.

5.3.2.2.1 Offspring, Survivor and Elite Count The new population size parameters are picked randomly from the interval between the parents' parameters as shown in Equations (5.11) and (5.12) The probability to perform a crossover on the population size is 0.5, since the population size is constant, crossover is applied to the offspring, survivor and elite counts altogether.

$$
\begin{aligned}
G = &\quad \{\text{Offspring count}, \text{Survivor count}, \text{Elite count}\} \\
\alpha_{u,g}: &\quad \text{Population parameter } g \text{ of parent } u && \forall u \in \{1,2\}, g \in G \\
\overline{\alpha}_{u,g}: &\quad \text{Population parameter } g \text{ of offspring } u && \forall u \in \{1,2\}, g \in G \\
\theta_g \sim &\quad \text{Uniform}(0,1), \text{Drawn separately for each } g && \forall g \in G
\end{aligned}
$$

$$\overline{\alpha}_{1,g} = \theta_g \times \alpha_{1,g} + (1 - \theta_g) \times \alpha_{2,g} \tag{5.11}$$

$$\overline{\alpha}_{2,g} = (1 - \theta_g) \times \alpha_{1,g} + \theta_g \times \alpha_{2,g} \tag{5.12}$$

5.3.2.2.2 Mutation Probability The crossover is performed on the mutation probabilities in a similar way, shown in Equations (5.13) and (5.14).

$$
\begin{aligned}
\sigma_u: &\quad \text{Mutation probability of parent } u && \forall u \in \{1,2\} \\
\overline{\sigma}_u: &\quad \text{Mutation probability of offspring } u && \forall u \in \{1,2\} \\
\theta = &\quad \text{Uniform}(0,1), \text{Drawn separately for each operation}
\end{aligned}
$$

$$\overline{\sigma}_1 = \theta \times \sigma_1 + (1 - \theta) \times \sigma_2 \tag{5.13}$$

$$\overline{\sigma}_2 = (1 - \theta) \times \sigma_1 + \theta \times \sigma_2 \tag{5.14}$$

5.3.2.2.3 Selection Operators The selection operators include the parent selector and the survivor selector, the same procedure is followed for each one. Since selectors have unique parameters with their own rules, parameters of the two selectors can only be combined when they are the same. Therefore, in case the parent algorithms have the same selector, their parameters recombined by the same combination operator used in mutation probability, as shown in equations (5.13) and (5.14). If the selectors are different, they are swapped without any modification, with a probability of 0.25.

The offspring are mutated by randomizing each algorithm parameter with a probability of 0.10, the randomization procedure is the same one used in the random setting generator. Unless they are randomized, parent and survivor selectors have a 25% chance of having their parameters randomized.

5.3.3 Terminating Condition

Once the offspring are mutated, they are merged with the selected elites and survivors to build the next generation, followed by the evaluation of the next generation. This process is repeated until the algorithm setting with the best fitness score, calculated as described in Section 5.3.1.1, does not change in the last 10 generations.

Chapter 6

Results

In this chapter, the performance of the proposed genetic algorithm and a comparison with the existing methodology in the literature are presented. The performance metric was obtained using the benchmark instances provided by Özbaygın et al.[19], instance-specific information can be found in Appendix A.

6.1 Fine-tuning Algorithm Results

The fine-tuning algorithm was implemented in Python 2.7.5 and run on a different computer than the model and genetic algorithm, with Intel Xeon CPU E5-2690 (56 cores) and 128 GB of RAM. It utilized parallel computation with 25-30 cores and it took around 128 hours to process each generation. 27 generations were processed before meeting the terminating condition that indicated a convergence. Compared to the first generation, the fine-tuning procedure was able to decrease the average gap, for the benchmark instances, from 0.3% to -18%, the standard deviation within the samples stayed roughly the same. The fitness value of the best algorithm setting and mean value of all of the solutions in the population, calculated using Equation 5.6, throughout the generations can be seen in Figure 6.1.

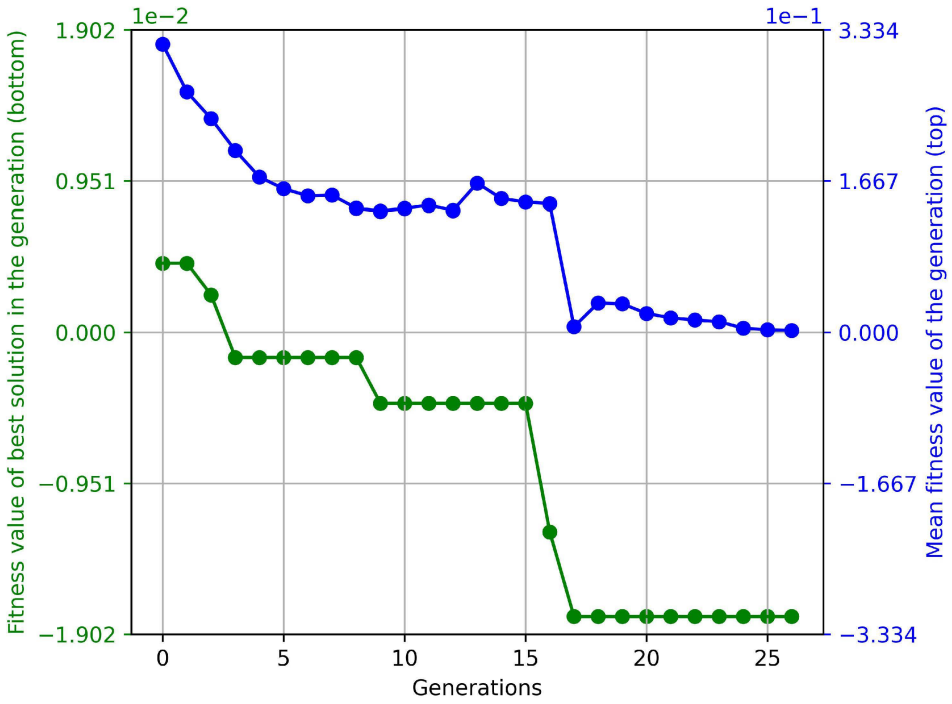

Figure 6.1: Fitness values over the generations

6.2 Solver-GA Results

In this section, results of the SGA as well as a comparative study with the results of the exact methodology in the literature is provided.

6.2.1 Removal of the Infeasible Nodes

The removal of the nodes that are infeasible to visit throughout the day, as determined by the Equations 5.7 and 5.8, had a significant impact on the processing speed of the genetic algorithm. On average, 40% of the customer nodes are infeasible to visit in an instance. Removal of these nodes inherently reduces the chromosome length which, in turn, speeds up the fitness value calculation (Algorithm 1) as it iterates through the chromosome. The improvement on the processing speed can be calculated as 40%, on average, as indicated by the decrease in the number of nodes. The customer and node count statistics for the

instances can be found in Appendix A.

6.2.2 Mathematical Model and Solver-GA

The mathematical model was implemented in and solved using CPLEX Studio 12.10, with a time limit of 3 hours and a branch and bound tree size limit of 400 GB. The tree was allowed to be written on HDD to avoid RAM limitations, which was an issue for instances with more than 60 customers where the solver usually run out of memory (16 GB) before meeting the time limit. The genetic algorithm was coded in Python 3.6.8. Both the model and algorithm were run on a computer with 16 GB of RAM, 500 GB of HDD, Intel Xeon Silver 4210 CPU (20 cores), the model was allowed to use all of the cores while the algorithm was allowed to use only one core (no parallel computation). The amount of total single-processor time allocated to the SGA is 300 seconds as it does not use parallel computation, the model, however, has 216,000 seconds of single-processor time due to utilizing 20 processors concurrently. Using the allocated single-processor times, it can be calculated that the processing power required by the model is 720 times of what is required by the algorithm to obtain the results represented in this thesis. To provide a better perspective on how allocating more time to the algorithm affects its performance, it was allowed to run for 5 and 15 minutes separately during the computational study.

Figure 6.2 shows the deviation of SGA output from the mathematical model with respect to the problem size in terms of number of nodes. The figures indicate that SGA is able to find solutions close to ones found by the model in smaller instances. The model is able to find significantly better solutions than SGA in the medium-sized instances because the problems are relatively small enough to be solved with a smaller optimality gap. In the large instances, however, the model struggles to reduce the optimality gap as expected. The benefit of increasing the algorithm time can also be seen from the figures.

The summary of results obtained by running SGA for 5 and 15 minutes on the problem instances can be seen on Tables 6.1 through 6.4. It is worth noting

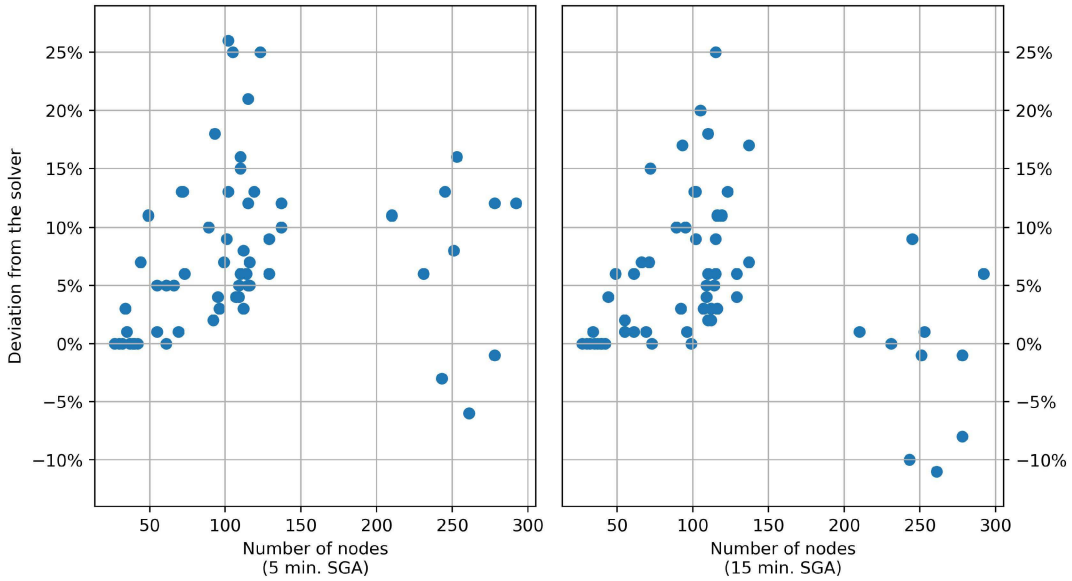

Figure 6.2: Gap vs. Node size distribution for 5 and 15 min. SGA

that the branch-and-price algorithm (B.P.) results are taken directly from the paper and were not run on the computer with which the model and SGA were taken from, instead they were run on an Intel Xeon E5-2650 cpu (16 cores). So it approximately has 57.5% of the computing power compared to the computer that was used to solve model. The time allocated for the B.P. is also different; instances with up to 60 customers were allowed to run for two hours while the larger ones were allowed for six hours. The efficiency of the branch-and-price algorithm is evident in the instances that it was able to solve as it was able to obtain solutions better than the model in 38 out of 60 instances, with less computational power. The difference in the computational power, however, does not have a significant affect when compared to the SGA, since it only requires a single core, the SGA can obtain approximately the same results independent of the number of cores available.

The detailed solution information can be found in Appendices B and C, which include the number of trucks used, the number of generations processed by the genetic algorithms, the objective value of the solutions found and the gap between the SGA and other methodologies for each instance.

Table 6.1: Summary of algorithm performance by number of customers (5 minutes)

$\|C\|$	Avg. gap model 5 min.	Avg. gap B.P. 5 min.
15	0% (min: 0%, max: 0.1%)	0% (min: 0%, max: 0.2%)
20	2.1% (min: 0%, max: 6.7%)	2.1% (min: 0%, max: 6.7%)
30	6.1% (min: 0.5%, max: 12.9%)	6.2% (min: 0.6%, max: 12.9%)
40	9.7% (min: 1.9%, max: 26.2%)	10.1% (min: 2%, max: 26.5%)
60	11.7% (min: 2.7%, max: 25.1%)	-1.6% (min: -64.1%, max: 25.8%)
120	6.8% (min: -6%, max: 16.3%)	-55.6% (min: -60.1%, max: -52.9%)
Total	7.5% (min: -6%, max: 26.2%)	-4.9% (min: -64.1%, max: 26.5%)

Table 6.2: Summary of algorithm performance by number of customers (15 minutes)

$\|C\|$	Avg. gap model 15 min.	Avg. gap B.P. 15 min.
15	0% (min: 0%, max: 0%)	0% (min: 0%, max: 0.2%)
20	1.1% (min: 0%, max: 4.3%)	1.1% (min: 0%, max: 4.4%)
30	4.7% (min: 0.1%, max: 14.8%)	4.8% (min: 0.1%, max: 15%)
40	7.8% (min: 0.4%, max: 20.3%)	8.2% (min: 0.5%, max: 20.9%)
60	9.9% (min: 1.7%, max: 24.6%)	-3.5% (min: -62.1%, max: 25%)
120	-1.5% (min: -10.9%, max: 8.6%)	-59.1% (min: -61.7%, max: -55.4%)
Total	4.9% (min: -10.9%, max: 24.6%)	-6.8% (min: -62.1%, max: 25%)

The results obtained for the SGA indicate that it is able to reach the near-optimal results for small instances with less than 20 customers. As the number of customers increase, the number of nodes that problem has inherently increases, which results in larger deviations from the optimal as expected. The increase in problem complexity becomes more noticeable as the number of customers increase, resulting in SGA to be able to reach solutions better than the model for instances with more than 120 customers.

Table 6.3: Summary of algorithm performance by number of nodes (5 minutes)

$\|N\|$	Avg. gap model 5 min.	Avg. gap B.P. 5 min.
[27,55]	2.2% (min:0%,max:11.4%)	2.2% (min:0%,max:11.5%)
[56,96]	6.8% (min:0.5%,max:18.1%)	6.9% (min:0.6%,max:18.2%)
[97,112]	10.9% (min:2.7%,max:26.2%)	11.5% (min:3.6%,max:26.5%)
[113,137]	11% (min:5%,max:25.1%)	-0.1% (min:-64.1%,max:25.8%)
[138,292]	6.8% (min:-6%,max:16.3%)	-55.6% (min:-60.1%,max:-52.9%)
Total	7.5% (min:-6%,max:26.2%)	-4.9% (min:-64.1%,max:26.5%)

Table 6.4: Summary of algorithm performance by number of nodes (15 minutes)

$\|N\|$	Avg. gap model 15 min.	Avg. gap B.P. 15 min.
[27,55]	1.1% (min:0%,max:5.6%)	1.1% (min:0%,max:5.7%)
[56,96]	6.5% (min:0.1%,max:16.9%)	6.6% (min:0.1%,max:16.9%)
[97,112]	7.6% (min:0.4%,max:20.3%)	8.2% (min:0.5%,max:20.9%)
[113,137]	9.6% (min:2.5%,max:24.6%)	-1.6% (min:-62.1%,max:25%)
[138,292]	-1.5% (min:-10.9%,max:8.6%)	-59.1% (min:-61.7%,max:-55.4%)
Total	4.9% (min:-10.9%,max:24.6%)	-6.8% (min:-62.1%,max:25%)

Chapter 7

Conclusions and Future Research Directions

Introduced with the development of the smart cars, the trunk delivery option for delivering packages to customers can decrease the total distance traveled, compared to the conventional home-delivery option. This option benefits from the customer movements by delivering their orders at locations closer to the trucks and depot. Even though a huge amount of data about the customer itineraries are required, there is a significant potential of improving the customer satisfaction and lowering the costs at the same time.

In Chapter 2, emergence of the problem in the industry, as well as its applications by the e-commerce companies are summarized. In Chapter 3, the up-to-date literature on the VRP-RDL, its sub-variants GVRP and VRP-TW, and the proposed solution methodologies are presented. In Chapter 4, a mixed-integer formulation of VRP-RDL from the literature and the underlying modelling assumptions, some inherited directly from the data available and others from the modelling approach, are described. In Chapter 5, the proposed solution generation procedure that is able to represent every feasible solution, a genetic algorithm based solution improvement methodology, a fine tuning algorithm to improve the parameters of the genetic algorithm based on a selected set of problem instances

are explained in detail. Finally, in Chapter 6, the results obtained from the proposed genetic algorithm and comparisons with the exact solution methodology in the literature are provided.

Considering that problem instances in real-life can include thousands of customers with even more delivery points, performance of the exact solution techniques in the literature are expected to be hindered by the computer limitations such as available memory and computational speed. Although its performance is surpassed by the exact methodologies in the small and medium sized instances, the proposed genetic algorithm is able to find better solutions in larger instances, with only 0.138% of the processing power allocated to the model. Moreover, other variants of the problem with additional constraints, such as split delivery option or a limit on the number of trucks, can easily be implemented into the proposed algorithm.

The proposed fine tuning framework for the genetic algorithms can be used on any genetic algorithm and is able to continuously find and improve algorithm settings based on a set of benchmark instances. In the literature, the process of deciding on the genetic algorithm settings are mostly done by intuition and trying out specific values for the parameters. The methodology we propose, however, achieves this by systematically generating new algorithm settings and is able to consider the settings that may otherwise be deemed bad quality or illogical.

The future research directions could be the dynamic version of the problem, which would be more realistic considering that some of the customer movements may change during the day. From the e-commerce companies' perspective, the most realistic dynamic version would include the information regarding the customer positions instantaneously throughout the day. This version of the problem, however, requires a significant amount of information, or assumptions, regarding the randomness of the customer movements. Another realistic perspective would include larger problem instances as a delivery person may make up to 200 deliveries per day on average [85]. It would also be beneficial to consider the multiple depot version, which would incorporate the order preparation perspective by deciding on customer-depot assignments. Other variants could be the ones that take

the randomness of the traffic, during different times of the day, and its effects on the distance and travel time into account, as most of the deliveries happen in the metropolitan areas. These variants would have to consider the effect of traffic not only on trucks, but also on the customer movements.

Bibliography

[1] J.-P. Rodrigue, *The geography of transport systems*. Routledge, 2020.

[2] T. G. Crainic, "City logistics," in *State-of-the-art decision-making tools in the information-intensive age*, pp. 181–212, INFORMS, 2008.

[3] G. Laporte, "The vehicle routing problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 3, pp. 345–358, 1992.

[4] DHL, "Dhl now delivering to trunks of vw cars: Dhl," Sep 2017. [Online; accessed 10-July-2021].

[5] D. Reyes, M. Savelsbergh, and A. Toriello, "Vehicle routing with roaming delivery locations," *Transportation Research Part C: Emerging Technologies*, vol. 80, pp. 71–91, 2017.

[6] S. B. Hepp, "Innovation in last mile delivery: meeting evolving customer demands: the case of in-car delivery," Master's thesis, Cátolica-Lisbon School of Business & Economics and Università Bocconi, 2018.

[7] K. Rooney, "Online shopping overtakes a major part of retail for the first time ever," Apr 2019.

[8] F. Ali, "Ecommerce trends amid coronavirus pandemic in charts," Mar 2021. [Online; accessed 10-July-2021].

[9] M. Haag and W. Hu, "1.5 million packages a day: The internet brings chaos to n.y. streets," Oct 2019.

[10] E. Cramer-Flood, "Global ecommerce 2020," Jun 2020.

[11] M. Hall, "Amazon.com," Oct 2017.

[12] F. Jordan, "Why is amazon so successful? – factors explained," Jan 2020.

[13] C. Emba, "Opinion — amazon key is silicon valley at its most out-of-touch," Oct 2017.

[14] D. Alaimo, "Almost 70% unwilling to use amazon key in-home delivery," Mar 2018.

[15] R. Kraus, "Amazon can now deliver packages to the trunk of your car," Apr 2018.

[16] B. F. Rubin, "Amazon key in-car delivers right to your car's trunk," Apr 2018.

[17] EuroCoc, "Amazon audi trunk delivery," May 2015. [Online; accessed 10-July-2021].

[18] Amazon, "Delivery information," 2011. [Online; accessed 10-July-2021].

[19] G. Ozbaygin, O. Ekin Karasan, M. Savelsbergh, and H. Yaman, "A branch-and-price algorithm for the vehicle routing problem with roaming delivery locations," *Transportation Research Part B: Methodological*, vol. 100, 2017.

[20] R. Baldacci, E. Bartolini, and G. Laporte, "Some applications of the generalized vehicle routing problem," *Journal of the operational research society*, vol. 61, no. 7, pp. 1072–1077, 2010.

[21] P. Toth and D. Vigo, "A heuristic algorithm for the symmetric and asymmetric vehicle routing problems with backhauls," *European Journal of Operational Research*, vol. 113, no. 3, pp. 528–543, 1999.

[22] G. Laporte and F. Semet, "Classical heuristics for the capacitated vrp," in *The vehicle routing problem*, pp. 109–128, SIAM, 2002.

[23] P. Toth and D. Vigo, *The vehicle routing problem*. SIAM, 2002.

[24] M. Desrochers, J. Desrosiers, and M. Solomon, "A new optimization algorithm for the vehicle routing problem with time windows," *Operations research*, vol. 40, no. 2, pp. 342–354, 1992.

[25] A. Moutaoukil, G. Neubert, and R. Derrouiche, "A comparison of homogeneous and heterogeneous vehicle fleet size in green vehicle routing problem," in *IFIP International Conference on Advances in Production Management Systems*, pp. 450–457, Springer, 2014.

[26] B. M. Baker and M. Ayechew, "A genetic algorithm for the vehicle routing problem," *Computers & Operations Research*, vol. 30, no. 5, pp. 787–800, 2003.

[27] S. N. Kumar and R. Panneerselvam, "A survey on the vehicle routing problem and its variants," *Intelligent Information Management*, 2012.

[28] L. Moccia, J.-F. Cordeau, and G. Laporte, "An incremental tabu search heuristic for the generalized vehicle routing problem with time windows," *Journal of the Operational Research Society*, vol. 63, no. 2, pp. 232–244, 2012.

[29] D. Dumez, F. Lehuédé, and O. Péton, "A large neighborhood search approach to the vehicle routing problem with delivery options," *Transportation Research Part B: Methodological*, vol. 144, pp. 103–132, 2021.

[30] M. Haag and W. Hu, "As online shopping surged, amazon planned its new york takeover," Mar 2021.

[31] G. Ozbaygin and M. Savelsbergh, "An iterative re-optimization framework for the dynamic vehicle routing problem with roaming delivery locations," *Transportation Research Part B: Methodological*, vol. 128, 2019.

[32] A. Lombard, S. Tamayo-Giraldo, and F. Fontane, "Vehicle routing problem with roaming delivery locations and stochastic travel times (vrprdl-s)," *Transportation Research Procedia*, vol. 30, pp. 167–177, 2018. EURO Mini Conference on "Advances in Freight Transportation and Logistics".

[33] A. Sampaio, J. Kinable, L. P. Veelenturf, and T. Van Woensel, "A scenario-based approach for the vehicle routing problem with roaming delivery locations under stochastic travel times," *Optim. Online*, 2019.

[34] J. H. Holland *et al.*, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[35] B. Waggener, W. N. Waggener, and W. M. Waggener, *Pulse code modulation techniques*. Springer Science & Business Media, 1995.

[36] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, pp. 707–710, Soviet Union, 1966.

[37] M. M. Flood, "The traveling-salesman problem," *Operations research*, vol. 4, no. 1, pp. 61–75, 1956.

[38] S. Mirjalili, *Genetic Algorithm*, pp. 43–55. Cham: Springer International Publishing, 2019.

[39] J. Hageman, R. Wehrens, H. Van Sprang, and L. Buydens, "Hybrid genetic algorithm–tabu search approach for optimising multilayer optical coatings," *Analytica Chimica Acta*, vol. 490, no. 1-2, pp. 211–222, 2003.

[40] M. A. Sahnehsaraei, M. J. Mahmoodabadi, M. Taherkhorsandi, K. K. Castillo-Villar, and S. M. Yazdi, "A hybrid global optimization algorithm: particle swarm optimization in association with a genetic algorithm," in *Complex System Modelling and Control Through Intelligent Soft Computations*, pp. 45–86, Springer, 2015.

[41] H. Yu, H. Fang, P. Yao, and Y. Yuan, "A combined genetic algorithm/simulated annealing algorithm for large scale system energy integration," *Computers & Chemical Engineering*, vol. 24, no. 8, pp. 2023–2035, 2000.

[42] M. Gong and Y.-H. Yang, "Multi-resolution stereo matching using genetic algorithm," in *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*, pp. 21–29, IEEE, 2001.

[43] A. Madureira, C. Ramos, and S. do Carmo Silva, "A coordination mechanism for real world scheduling problems using genetic algorithms," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, vol. 1, pp. 175–180, IEEE, 2002.

[44] F. H.-F. Leung, H.-K. Lam, S.-H. Ling, and P. K.-S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Transactions on Neural networks*, vol. 14, no. 1, pp. 79–88, 2003.

[45] K. Otubamowo, T. Egunjobi, and A. Adewole, "A comparative study of simulated annealing and genetic algorithm for solving the travelling salesman problem," 2012.

[46] A. D. Lidbe, A. M. Hainen, and S. L. Jones, "Comparative study of simulated annealing, tabu search, and the genetic algorithm for calibration of the microsimulation model," *Simulation*, vol. 93, no. 1, pp. 21–33, 2017.

[47] M. Hasan, T. Alkhamis, and J. Ali, "A comparison between simulated annealing, genetic algorithm and tabu search methods for the unconstrained quadratic pseudo-boolean function," *Computers & industrial engineering*, vol. 38, no. 3, pp. 323–340, 2000.

[48] K. Mak and Z. Guo, "A genetic algorithm for vehicle routing problems with stochastic demand and soft time windows," in *Proceedings of the 2004 IEEE Systems and Information Engineering Design Symposium, 2004.*, pp. 183–190, IEEE, 2004.

[49] D. M. Pierre and N. Zakaria, "Stochastic partially optimized cyclic shift crossover for multi-objective genetic algorithms for the vehicle routing problem with time-windows," *Applied Soft Computing*, vol. 52, pp. 863–876, 2017.

[50] J. Berger and M. Barkaoui, "A parallel hybrid genetic algorithm for the vehicle routing problem with time windows," *Computers & operations research*, vol. 31, no. 12, pp. 2037–2053, 2004.

[51] K. Tan, L. Lee, Q. Zhu, and K. Ou, "Heuristic methods for vehicle routing problem with time windows," *Artificial Intelligence in Engineering*, vol. 15, no. 3, pp. 281–295, 2001.

[52] K. Q. Zhu, "A diversity-controlling adaptive genetic algorithm for the vehicle routing problem with time windows," in *Proceedings. 15th IEEE International Conference on Tools with Artificial Intelligence*, pp. 176–183, IEEE, 2003.

[53] S. Jung and B. R. Moon, "A hybrid genetic algorithm for the vehicle routing problem with time windows.," in *GECCO*, pp. 1309–1316, 2002.

[54] B. Ombuki, B. J. Ross, and F. Hanshar, "Multi-objective genetic algorithms for vehicle routing problem with time windows," *Applied Intelligence*, vol. 24, no. 1, pp. 17–30, 2006.

[55] S. Jung and A. Haghani, "Genetic algorithm for the time-dependent vehicle routing problem," *Transportation Research Record*, vol. 1771, no. 1, pp. 164–171, 2001.

[56] H. Nazif and L. S. Lee, "Optimized crossover genetic algorithm for vehicle routing problem with time windows," *American journal of applied sciences*, vol. 7, no. 1, p. 95, 2010.

[57] Y. Chang and L. Chen, "Solve the vehicle routing problem with time windows via a genetic algorithm," *Discrete and continuous dynamical systems supplement*, pp. 240–249, 2007.

[58] S.-H. Xu, J.-P. Liu, F.-H. Zhang, L. Wang, and L.-J. Sun, "A combination of genetic algorithm and particle swarm optimization for vehicle routing problem with time windows," *Sensors*, vol. 15, no. 9, pp. 21033–21053, 2015.

[59] J. Xu, G. Goncalves, and T. Hsu, "Genetic algorithm for the vehicle routing problem with time windows and fuzzy demand," in *2008 IEEE Congress on*

Evolutionary Computation (IEEE World Congress on Computational Intelligence), pp. 4125–4129, IEEE, 2008.

[60] J. Berger, M. Salois, and R. Begin, "A hybrid genetic algorithm for the vehicle routing problem with time windows," in *Conference of the canadian society for computational studies of intelligence*, pp. 114–127, Springer, 1998.

[61] K. Wang, S. Lan, and Y. Zhao, "A genetic-algorithm-based approach to the two-echelon capacitated vehicle routing problem with stochastic demands in logistics service," *Journal of the Operational Research Society*, vol. 68, no. 11, pp. 1409–1421, 2017.

[62] K. Ghoseiri and S. F. Ghannadpour, "Multi-objective vehicle routing problem with time windows using goal programming and genetic algorithm," *Applied Soft Computing*, vol. 10, no. 4, pp. 1096–1107, 2010.

[63] G. B. Alvarenga, G. R. Mateus, and G. De Tomi, "A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows," *Computers & Operations Research*, vol. 34, no. 6, pp. 1561–1584, 2007.

[64] P. Li, J. He, D. Zheng, Y. Huang, and C. Fan, "Vehicle routing problem with soft time windows based on improved genetic algorithm for fruits and vegetables distribution," *Discrete Dynamics in Nature and Society*, vol. 2015, 2015.

[65] J.-Y. Potvin and S. Bengio, "The vehicle routing problem with time windows part ii: genetic search," *INFORMS journal on Computing*, vol. 8, no. 2, pp. 165–172, 1996.

[66] S. R. Thangiah, K. E. Nygard, and P. L. Juell, "Gideon: A genetic algorithm system for vehicle routing with time windows," in *Proceedings The Seventh IEEE Conference on Artificial Intelligence Application*, pp. 322–323, IEEE Computer Society, 1991.

[67] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins, "A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle

routing problems with time-windows," *Computers & operations research*, vol. 40, no. 1, pp. 475–489, 2013.

[68] G. Ghiani and G. Improta, "An efficient transformation of the generalized vehicle routing problem," *European Journal of Operational Research*, vol. 122, no. 1, pp. 11–17, 2000.

[69] P. Pop, O. Matei, and H. Valean, "An efficient hybrid soft computing approach to the generalized vehicle routing problem," in *Soft Computing Models in Industrial and Environmental Applications, 6th International Conference SOCO 2011*, pp. 281–289, Springer, 2011.

[70] P. C. Pop, O. Matei, and C. P. Sitar, "An improved hybrid algorithm for solving the generalized vehicle routing problem," *Neurocomputing*, vol. 109, pp. 76–83, 2013. New trends on Soft Computing Models in Industrial and Environmental Applications.

[71] Y. Deng, Y. Liu, and D. Zhou, "An improved genetic algorithm with initial population strategy for symmetric tsp," *Mathematical Problems in Engineering*, vol. 2015, 2015.

[72] J. Yang and C. K. Soh, "Structural optimization by genetic algorithms with tournament selection," *Journal of Computing in Civil Engineering*, vol. 11, no. 3, pp. 195–200, 1997.

[73] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 6, pp. 2193–2196, 2012.

[74] H. Muhlenbein, "The breeder genetic algorithm-a provable optimal search algorithm and its application," in *IEE Colloquium on Applications of Genetic Algorithms*, pp. 5–1, IET, 1994.

[75] B. A. Julstrom, "It's all the same to me: Revisiting rank-based probabilities and tournaments," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 2, pp. 1501–1505, IEEE, 1999.

[76] I. Loshchilov, M. Schoenauer, and M. Sebag, "Not all parents are equal for mo-cma-es," in *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 31–45, Springer, 2011.

[77] C.-Y. Lee, "Entropy-boltzmann selection in the genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 33, no. 1, pp. 138–149, 2003.

[78] P. Kora and P. Yadlapalli, "Crossover operators in genetic algorithms: A review," *International Journal of Computer Applications*, vol. 162, no. 10, 2017.

[79] P. W. Poon and J. N. Carter, "Genetic algorithm crossover operators for ordering applications," *Computers & Operations Research*, vol. 22, no. 1, pp. 135–147, 1995.

[80] W. M. Spears and K. A. De Jong, "An analysis of multi-point crossover," in *Foundations of genetic algorithms*, vol. 1, pp. 301–315, Elsevier, 1991.

[81] C. L. Bridges and D. E. Goldberg, "An analysis of reproduction and crossover in a binary-coded genetic algorithm," *Grefenstette*, vol. 878, pp. 9–13, 1987.

[82] X.-B. Hu and E. Di Paolo, "An efficient genetic algorithm with uniform crossover for air traffic control," *Computers & Operations Research*, vol. 36, no. 1, pp. 245–259, 2009.

[83] T.-P. Hong and H.-S. Wang, "A dynamic mutation genetic algorithm," in *1996 IEEE International Conference on Systems, Man and Cybernetics. Information Intelligence and Systems (Cat. No. 96CH35929)*, vol. 3, pp. 2000–2005, IEEE, 1996.

[84] A. A. Radwan, B. A. A. Latef, A. M. A. Ali, and O. A. Sadek, "Using genetic algorithm to improve information retrieval systems," *World Academy of Science and Engineering Technology*, vol. 17, no. 2, pp. 6–13, 2006.

[85] BBC, "A day in the life of an amazon delivery driver," Nov 2016.

Appendix A

Benchmark Instances

Table A.1: Benchmark Instances

Instance	FGA instance	C	N	# of feas. nodes	Infeas. nodes %
1	✓	15	62	39	37%
2	✓	15	57	32	44%
3	✓	15	52	37	29%
4	✓	15	50	27	46%
5	✓	15	52	30	42%
6	✓	20	66	40	39%
7	✗	20	68	34	50%
8	✗	20	80	44	45%
9	✗	20	76	42	45%
10	✗	20	63	35	44%
11	✗	30	103	61	41%
12	✗	30	113	69	39%
13	✗	30	118	71	40%
14	✗	30	107	61	43%
15	✗	30	124	72	42%
16	✗	30	119	73	39%
17	✗	30	130	66	49%
18	✗	30	106	49	54%
19	✗	30	98	55	44%
20	✗	30	75	55	27%
21	✗	60	225	137	39%
22	✗	60	213	119	44%
23	✗	60	235	129	45%

Table A.2: Benchmark Instances continued

Instance	FGA instance	C	N	# of feas. nodes	Infeas. nodes %
24	✗	60	229	123	46%
25	✗	60	234	137	41%
26	✓	60	225	115	49%
27	✓	60	208	115	45%
28	✓	60	219	129	41%
29	✓	60	226	112	50%
30	✓	60	238	115	52%
31	✓	120	461	278	40%
32	✓	120	441	251	43%
33	✓	120	471	292	38%
34	✓	120	455	278	39%
35	✓	120	451	261	42%
36	✓	120	422	210	50%
37	✓	120	467	245	48%
38	✓	120	422	243	42%
39	✓	120	428	231	46%
40	✓	120	462	253	45%
41_v1	✓	40	160	95	41%
41_v2	✓	40	160	105	34%
42_v1	✓	40	158	110	30%
42_v2	✗	40	158	112	29%
43_v1	✓	40	163	101	38%
43_v2	✗	40	163	109	33%
44_v1	✓	40	142	93	35%
44_v2	✗	40	142	102	28%
45_v1	✓	40	145	109	25%
45_v2	✗	40	145	116	20%
46_v1	✓	40	174	96	45%
46_v2	✗	40	174	110	37%
47_v1	✗	40	168	107	36%
47_v2	✓	40	168	110	35%
48_v1	✗	40	154	89	42%
48_v2	✗	40	154	99	36%
49_v1	✗	40	153	92	40%
49_v2	✗	40	153	102	33%
50_v1	✗	40	162	114	30%
50_v2	✓	40	162	116	28%
Average					40%

Appendix B

Solver-GA Results

Table B.1: Model, BP and SGA results

Instance	Y_I	(Z_R)	Y_{BP}	Y_{SGA-5}	Y_{SGA-15}	G_{SGA-5}	G_{SGA-15}
1	901	(0%)	901	901	901	53,346	152,400
2	1,286	(0%)	1,286	1,286	1,286	54,310	163,430
3	992	(0%)	991	992	992	55,249	160,992
4	1,064	(0%)	1,062	1,064	1,064	55,927	169,756
5	1,832	(0%)	1,832	1,832	1,832	56,169	163,055
6	1,293	(0%)	1,294	1,294	1,293	45,399	127,205
7	1,155	(0%)	1,155	1,185	1,167	45,093	137,031
8	1,456	(0%)	1,455	1,553	1,519	40,485	124,804
9	1,260	(0%)	1,260	1,260	1,260	43,282	129,683
10	1,687	(0%)	1,684	1,705	1,687	44,693	133,301
11	1,925	(0%)	1,922	1,934	1,952	29,027	81,769
12	2,328	(0%)	2,324	2,354	2,349	27,037	78,345
13	1,747	(0%)	1,747	1,972	1,867	31,333	77,784
14	1,273	(0%)	1,273	1,343	1,353	31,661	87,502
15	1,696	(0%)	1,694	1,912	1,947	27,244	80,510
16	1,938	(0%)	1,938	2,054	1,941	27,984	79,877
17	1,968	(0%)	1,965	2,058	2,112	27,074	83,229
18	1,829	(0%)	1,827	2,037	1,931	32,032	92,771
19	2,086	(0%)	2,083	2,193	2,131	29,169	87,678
20	1,824	(13,3%)	1,822	1,848	1,843	31,559	93,826
21	3,767	(43%)	3,761	4,212	4,021	14,272	42,128
22	2,831	(36,8%)	2,828	3,193	3,138	14,672	44,278
23	4,449	(36,3%)	4,440	4,715	4,726	14,486	42,039

Table B.2: Model, BP and SGA results continued

Instance	Y_I	(Z_R)	Y_{BP}	Y_{SGA-5}	Y_{SGA-15}	G_{SGA-5}	G_{SGA-15}
24	3,399	(31,6%)	3,378	4,251	3,841	14,679	43,675
25	3,176	(31,5%)	9,784	3,509	3,709	14,397	43,272
26	4,544	(36,7%)	4,536	4,786	4,943	14,861	43,801
27	2,873	(30,9%)	2,865	3,479	3,581	15,599	45,940
28	4,177	(39,6%)	9,450	4,565	4,347	14,303	41,475
29	3,995	(38,2%)	3,964	4,105	4,062	15,241	44,625
30	4,114	(32,4%)	4,107	4,609	4,371	15,318	44,274
31	7,689	(73,6%)	18,142	7,607	7,086	6,905	20,086
32	8,007	(74,2%)	19,514	8,649	7,898	7,238	21,104
33	7,567	(69,7%)	18,008	8,481	8,029	6,703	19,498
34	8,359	(74,9%)	19,880	9,367	8,235	6,926	20,185
35	9,011	(79,4%)	19,196	8,474	8,030	7,082	20,708
36	9,004	(75,8%)	21,772	9,979	9,104	7,854	22,942
37	7,063	(70,8%)	20,010	7,988	7,673	7,288	21,405
38	8,961	(76,5%)	20,032	8,662	8,023	7,327	21,488
39	8,683	(76,3%)	20,136	9,203	8,679	7,499	21,668
40	7,687	(73%)	20,042	8,937	7,772	7,334	20,929
41_v1	3,204	(40%)	3,203	3,336	3,515	20,798	62,209
41_v2	2,142	(23,5%)	2,133	2,682	2,578	20,122	61,307
42_v1	2,804	(41,6%)	2,799	3,262	2,971	20,116	55,260
42_v2	2,006	(24,4%)	1,946	2,172	2,068	20,869	57,919
43_v1	2,609	(38,7%)	2,607	2,837	2,945	20,648	62,431
43_v2	1,970	(23,7%)	1,966	2,078	2,061	19,681	58,601
44_v1	2,262	(37,4%)	2,261	2,671	2,644	20,571	59,378
44_v2	1,614	(24,6%)	1,610	2,037	1,832	20,418	60,424
45_v1	3,225	(45,8%)	3,217	3,342	3,359	19,100	55,927
45_v2	2,483	(35,4%)	2,478	2,650	2,748	19,573	55,675
46_v1	2,806	(23,2%)	2,805	2,896	2,835	20,542	62,453
46_v2	2,472	(15,5%)	2,469	2,631	2,525	19,072	58,355
47_v1	3,390	(41,7%)	3,339	3,514	3,486	19,878	59,231
47_v2	1,950	(25,9%)	1,946	2,234	2,303	21,263	55,725
48_v1	3,333	(37%)	3,325	3,682	3,671	20,801	63,068
48_v2	2,382	(21,6%)	2,380	2,556	2,393	20,521	63,636

Table B.3: Model, BP and SGA results continued

Instance	Y_I	(Z_R)	Y_{BP}	Y_{SGA-5}	Y_{SGA-15}	G_{SGA-5}	G_{SGA-15}
49_v1	3,540	(38,6%)	3,534	3,606	3,629	21,753	60,490
49_v2	2,492	(10,6%)	2,492	2,819	2,725	21,502	57,256
50_v1	2,758	(39%)	2,752	2,926	2,895	18,508	58,129
50_v2	2,446	(32,1%)	2,443	2,569	2,508	21,013	59,905

Y_I, Y_{BP}, Y_{SGA-5}, Y_{SGA-15}: Objective of the solutions found by model, branch-and-price, SGA 5 minutes, SGA 15 minutes respectively.
Z_R: Gap between Y_I and the relaxed model objective at the end of 3 hours.
G_{SGA-5}, G_{SGA-15}: Total number of generations processed by SGA with an algorithm time of 5 and 15 minutes respectively.

Appendix C

Solver-GA Solution Results

Table C.1: Solution and objective comparisons

Instance	Z_{I-S5}	Z_{I-S15}	Z_{BP-S5}	Z_{BP-S15}	$\|R_I\|$	$\|R_{S-5}\|$	$\|R_{S-15}\|$
1	0%	0%	0%	0%	4	4	4
2	0%	0%	0%	0%	6	5	5
3	0.1%	0%	0.1%	0.1%	5	4	4
4	0%	0%	0.2%	0.2%	5	5	5
5	0%	0%	0%	0%	7	6	6
6	0%	0%	0%	0%	6	5	5
7	2.7%	1%	2.6%	1%	4	5	4
8	6.7%	4.3%	6.7%	4.4%	6	6	6
9	0%	0%	0%	0%	5	5	5
10	1.1%	0%	1.3%	0.2%	7	7	8
11	0.5%	1.4%	0.6%	1.6%	8	7	7
12	1.1%	0.9%	1.3%	1.1%	9	9	8
13	12.9%	6.9%	12.9%	6.9%	7	8	8
14	5.5%	6.3%	5.5%	6.3%	7	7	6
15	12.8%	14.8%	12.9%	15%	6	8	8
16	6%	0.1%	6%	0.1%	7	8	7
17	4.6%	7.3%	4.7%	7.5%	9	8	9
18	11.4%	5.6%	11.5%	5.7%	7	8	7
19	5.1%	2.2%	5.3%	2.3%	8	8	8
20	1.3%	1.1%	1.4%	1.2%	6	7	7
21	11.8%	6.7%	12%	6.9%	14	15	15
22	12.8%	10.9%	12.9%	11%	11	13	12
23	6%	6.2%	6.2%	6.4%	16	18	18

Table C.2: Solution and objective comparisons continued

Instance	Z_{I-S5}	Z_{I-S15}	Z_{BP-S5}	Z_{BP-S15}	$\|R_I\|$	$\|R_{S-5}\|$	$\|R_{S-15}\|$
24	25.1%	13%	25.8%	13.7%	12	16	14
25	10.5%	16.8%	-64.1%	-62.1%	11	14	15
26	5.3%	8.8%	5.5%	9%	17	17	18
27	21.1%	24.6%	21.4%	25%	10	13	13
28	9.3%	4.1%	-51.7%	-54%	14	16	15
29	2.7%	1.7%	3.6%	2.5%	16	15	14
30	12%	6.3%	12.2%	6.4%	15	16	16
31	-1.1%	-7.8%	-58.1%	-60.9%	38	41	34
32	8%	-1.4%	-55.7%	-59.5%	36	46	37
33	12.1%	6.1%	-52.9%	-55.4%	37	43	36
34	12.1%	-1.5%	-52.9%	-58.6%	35	44	35
35	-6%	-10.9%	-55.9%	-58.2%	39	47	37
36	10.8%	1.1%	-54.2%	-58.2%	35	42	34
37	13.1%	8.6%	-60.1%	-61.7%	30	40	35
38	-3.3%	-10.5%	-56.8%	-60%	35	46	35
39	6%	0%	-54.3%	-56.9%	37	47	39
40	16.3%	1.1%	-55.4%	-61.2%	33	44	36
41_v1	4.1%	9.7%	4.2%	9.7%	10	11	12
41_v2	25.2%	20.3%	25.7%	20.9%	8	11	10
42_v1	16.3%	5.9%	16.5%	6.1%	9	11	10
42_v2	8.3%	3.1%	11.6%	6.3%	8	9	7
43_v1	8.8%	12.9%	8.8%	13%	9	10	10
43_v2	5.5%	4.6%	5.7%	4.8%	8	9	9
44_v1	18.1%	16.9%	18.2%	16.9%	8	9	10
44_v2	26.2%	13.5%	26.5%	13.8%	7	9	7
45_v1	3.6%	4.2%	3.9%	4.4%	11	11	11
45_v2	6.7%	10.7%	6.9%	10.9%	9	10	9
46_v1	3.2%	1%	3.2%	1.1%	9	9	9
46_v2	6.4%	2.1%	6.6%	2.3%	9	10	9
47_v1	3.7%	2.8%	5.3%	4.4%	11	11	11
47_v2	14.6%	18.1%	14.8%	18.3%	7	8	9
48_v1	10.5%	10.1%	10.7%	10.4%	11	12	12
48_v2	7.3%	0.4%	7.4%	0.5%	8	9	8

Table C.3: Solution and objective comparisons continued

Instance	Z_{I-S5}	Z_{I-S15}	Z_{BP-S5}	Z_{BP-S15}	$\|R_I\|$	$\|R_{S-5}\|$	$\|R_{S-15}\|$
49_v1	1.9%	2.5%	2%	2.7%	11	11	11
49_v2	13.1%	9.4%	13.1%	9.4%	8	11	10
50_v1	6.1%	5%	6.3%	5.2%	10	10	9
50_v2	5%	2.5%	5.2%	2.7%	8	9	9
Average	7.5%	4.9%	-4.9%	-6.8%	13.32	15.38	13.78

Z_{I-S5}, Z_{I-S15}, Z_{BP-S5}, Z_{BP-S15}: Gap between the objectives of model (I), branch-and-price algorithm (BP) and the corresponding SGA ($S5$, $S15$: SGA 5 and 15 minutes). i.e. $Z_{I-S5} = \frac{Y_{SGA-5}}{Y_I} - 1$.
$|R_I|$, $|R_{S-5}|$, $|R_{S-15}|$: Total number of trucks used by the solution of model, SGA 5 minutes and 15 minutes, respectively.