

A Metaheuristic Approach to Crowdsourced Vehicle Routing

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Logic and Computation

eingereicht von

Grace Lydia Longo

Matrikelnummer 11838518

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Priv. Doz. Dr. Nysret Musliu

Mitwirkung: Dr.in techn. Shqiponja Ahmetaj, M.Sc.A. MSc

Wien, 7. Dezember 2022


Grace Lydia Longo


Nysret Musliu

A Metaheuristic Approach to Crowdsourced Vehicle Routing

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Logic and Computation

by

Grace Lydia Longo

Registration Number 11838518

to the Faculty of Informatics

at the TU Wien

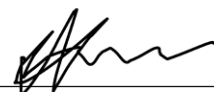
Advisor: Priv. Doz. Dr. Nysret Musliu

Assistance: Dr.in techn. Shqiponja Ahmetaj, M.Sc.A. MSc

Vienna, 7th December, 2022



Grace Lydia Longo



Nysret Musliu

Erklärung zur Verfassung der Arbeit

Grace Lydia Longo

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 7. Dezember 2022



Grace Lydia Longo

Acknowledgements

My thanks to my advisor Nysret Musliu and co-advisor Shqiponja Ahmetaj for their supervision, advice, expertise, and input. I would also like to further thank Shqiponja for her problem definition and Constraint Programming work that made this thesis possible.

I would also like to extend my deepest gratitude to Joseph Longo for his advise, proof-reading, and support. I wish to express my deepest appreciation to Eve Longo for her valuable advice on scientific writing. Thanks to Tamara Drucks for answering my many questions on the master's thesis process and the positive feedback and to Alicia Pretorius for her assistance in handing in the thesis.

I wish to also thank Claire, Seth, and Isaac Longo for their support and confidence in my work.

Abstract

Increased demand on the package delivery industry is prompting companies to make use of crowdsourced delivery drivers. In this industry, when crowdsourced drivers are used the drivers need to be quickly provided with an optimal delivery route. This thesis addresses a new, real-world, crowdsourcing variant of the Vehicle Routing Problem, with deviation constraints and both heterogeneous and homogeneous capacity constraints using ad-hoc drivers. Previously, the problem has only been solved with exact techniques. Although exact techniques guarantee exact optimal solutions, they can have very long running times making them impractical in a fast paced, real-world situation. For this reason, this thesis explores a metaheuristic approach to solving the crowdsourced VRP, where solutions are not guaranteed to be exact, but can be close to exact and the running times are much shorter, thus practical in a real-world setting.

In this thesis, we develop two Memetic Algorithms (HA1 and HA2) by combining a Genetic Algorithm with a Randomized Hill Climbing Local Search. The algorithms aim is the problem's objective of minimizing total deviations of all the drivers while satisfying the problem's constraints. Problem-specific crossover and mutation operations are developed, and two variants of the crossover operation are explored. The results of the algorithms are empirically evaluated. The algorithms best fitness values and running times are compared to results of a Constraint Programming solver and the algorithms are compared against each other.

One Memetic Algorithm in particular, HA1, provided positive results when evaluated. This algorithm is able to provide feasible solutions to instances with up to 135 packages with average running times of only a few minutes. On small instances the Memetic and Genetic Algorithms found feasible, but not exact, solutions, compared to the exact solutions of the Constraint Programming solver. Both Memetic Algorithms found more feasible and better results than the Genetic Algorithm, with HA1 performing the best in this area. The results found with a Memetic approach are practical for real-world problems since results that are both feasible and practical are provided within very short running times.

Kurzfassung

Die steigende Nachfrage in der Paketzustellbranche veranlasst die Unternehmen, Crowdsourced-Delivery-Fahrer einzusetzen. Wenn in dieser Branche Crowdsourcing-Fahrer eingesetzt werden, muss den Fahrern schnell eine optimale Lieferroute zur Verfügung gestellt werden. Diese Arbeit befasst sich mit einer neuen, realen Crowdsourcing-Variante des Fahrzeug-Routing-Problems mit Abweichungsbeschränkungen und sowohl heterogenen als auch homogenen Kapazitätsbeschränkungen unter Verwendung von Ad-hoc-Fahrern. Bislang wurde das Problem nur mit exakten Verfahren gelöst. Exakte Verfahren garantieren zwar exakte optimale Lösungen, können aber sehr lange Laufzeiten haben, was sie in einer schnelllebigen, realen Situation unpraktisch macht. Aus diesem Grund wird in dieser Arbeit ein metaheuristischer Ansatz zur Lösung des Crowdsourced VRP erforscht, bei dem die Lösungen zwar nicht garantiert exakt sind, aber nahe an der Exaktheit liegen können und die Laufzeiten viel kürzer sind, was in einer realen Umgebung praktisch ist.

In dieser Arbeit entwickeln wir zwei memetische Algorithmen (HA1 und HA2), indem wir einen genetischen Algorithmus mit einer lokalen Suche (Randomized Hill Climbing) kombinieren. Ziel der Algorithmen ist es, die Gesamtabweichungen aller Fahrer zu minimieren und gleichzeitig die Randbedingungen des Problems zu erfüllen. Es werden problemspezifische Crossover- und Mutationsoperationen entwickelt und zwei Varianten der Crossover-Operation untersucht. Die Ergebnisse der Algorithmen werden empirisch ausgewertet. Die besten Fitnesswerte und Laufzeiten der Algorithmen werden mit den Ergebnissen eines Constraint Programming Solvers verglichen und die Algorithmen werden miteinander verglichen.

Insbesondere ein memetischer Algorithmus, HA1, lieferte bei der Auswertung positive Ergebnisse. Dieser Algorithmus ist in der Lage, machbare Lösungen für Instanzen mit bis zu 135 Paketen bei durchschnittlichen Laufzeiten von nur wenigen Minuten zu liefern. Bei kleinen Instanzen fanden der memetische und der genetische Algorithmus praktikable, aber nicht exakte Lösungen, verglichen mit den exakten Lösungen des Constraint Programming Solvers. Beide memetischen Algorithmen fanden mehr realisierbare und bessere Ergebnisse als der genetische Algorithmus, wobei HA1 in diesem Bereich am besten abschnitt. Die mit dem memetischen Ansatz gefundenen Ergebnisse sind für reale Probleme praktikabel, da die Ergebnisse, die sowohl machbar als auch praktikabel sind, innerhalb sehr kurzer Laufzeiten geliefert werden.

Contents

| | |
|---|-----------|
| Abstract | ix |
| Kurzfassung | xi |
| Contents | xiii |
| 1 Introduction | 1 |
| 1.1 Motivation | 1 |
| 1.2 Aims of This Thesis | 3 |
| 1.3 Contributions of This Thesis | 3 |
| 1.4 Structure Of This Thesis | 4 |
| 2 Problem Statement and Related Work | 7 |
| 2.1 Problem Statement | 7 |
| 2.2 Related Work | 10 |
| 3 The Algorithms | 15 |
| 3.1 Genetic Algorithm | 15 |
| 3.2 Memetic Algorithm | 24 |
| 3.3 Local Search | 25 |
| 4 Experiments and Results | 29 |
| 4.1 Parameter Tuning | 30 |
| 4.2 Comparison of Algorithms | 32 |
| 5 Conclusion | 41 |
| 5.1 Summary of Work | 41 |
| 5.2 Future Work | 42 |
| A | 43 |
| List of Figures | 47 |
| List of Tables | 49 |
| | xiii |

Bibliography

Introduction

1.1 Motivation

In recent years, there has been increasing demand on the package delivery industry. Largely due to online sales, there is an ever-increasing number of packages needing to be delivered. The demand is exacerbated by increasing customer expectations for shorter delivery times. This puts a considerable strain on the package delivery industry. The traditional method of package delivery involves the use of a standard fleet of trucks with the single purpose of making deliveries and returning to their distribution center. This method has worked well in the past but now struggles to keep pace with the new demands of the industry. The traditional method is not only costly for the delivery company but also has a large negative impact on the environment due to the emissions produced by the delivery trucks. Due to these drawbacks, many companies including Amazon [1, 2] and Walmart [3], are turning to crowdshipping as a solution.

Crowdsourcing refers to the use of many small contributions by people volunteering to complete a larger task or solve a problem. Wikipedia is a prime example of crowdsourcing. People from all over the world contribute to each Wikipedia article by writing, editing, and researching Wikipedia content [4]. This leads to a massive collection of knowledge created and maintained with the help of a large number of people, all making small contributions to the whole. Crowdsourcing may also be leveraged in the delivery industry. Crowdsourced deliveries (known as crowdshipping) utilize everyday people who agree to deliver a few packages on their way to their own destination – e.g. their place of work. They are often compensated monetarily or otherwise for their time and effort. The crowdshipping approach has become a popular solution to the increased demand on the delivery industry. A single driver may deliver only a few packages but collectively hundreds of packages may be delivered in this manner, with only a small contribution from each driver.

In this thesis, we will tackle a real-world, crowdsourced Vehicle Routing Problem, where ad-hoc drivers deliver packages on their way to their own destination. Each driver specifies the maximum volume they are willing to transport, and the vehicles have a maximum number of packages they can transport. The drivers also specify the maximum distance they are willing to deviate from their original route. The objective of the problem is to minimize the total distance that all the vehicles deviate from their original route. This new, real-world problem was defined, formalized, and studied using Constraint Programming methods in 2020 by [5] at the Technische Universität Wien. The problem is a variant of the NP-hard Vehicle Routing Problem. So far the problem has only been solved using exact methods and no metaheuristic approach has been suggested.

Given enough time, the Constraint Programming method guarantees exact optimal solutions. However, exact methods for NP-hard problems such as this have the drawback of having impractically long running times. This is especially true for large instances. Since the problem is NP-hard, as the size of the instances grow, the running times of exact methods grow greatly. This can be impractical in real-world situations when a solution is needed quickly. Even the fastest exact methods have this issue. However, an exact solution is often not needed and an approximate solution is acceptable if it can be found faster. In the world of NP-hard problems, there is often a trade-off between accuracy and speed; an extremely fast algorithm at the cost of some accuracy is often ideal in real-world situations. Thus, metaheuristics are often an excellent solution to these types of problems. Metaheuristics do not guarantee precisely accurate solutions but they tend to be considerably faster than exact methods. In our case, we have a real-world package delivery problem where companies need to provide routes to drivers quickly in order for packages to be delivered daily. In this case, waiting for an exact optimal solution could take a considerable amount of time, especially when hundreds of packages require delivery. Because of the demand on the industry to deliver a large number of packages within increasing short delivery times, it may not be efficient to use exact algorithms since waiting for solutions could take hours when such a large number of packages is being considered. Hence metaheuristics are needed in this area; providing a good, but not perfect, set of routes within a few minutes is more practical than providing the best routes within several hours.

For this reason, we tackle the problem using a metaheuristic approach. The aim is to provide an approach that can be used in real-world situations and is practical for use in the industry. We use a Genetic and Memetic approach to create metaheuristic solutions to the problem. A Genetic Algorithm is a metaheuristic roughly based on the evolutionary process, where good solutions are chosen to create a new population of solutions that are more adapted to solve the problem. [6]. A Memetic Algorithm is a Genetic Algorithm combined with a Local Search procedure to make use of the beneficial aspects of the Local Search [7]. A Genetic Algorithm on its own may not provide the best results for Vehicle Routing Problems, but when combining a Genetic Algorithm with a Local Search procedure we expect fast and close to accurate results to this problem.

1.2 Aims of This Thesis

In this thesis, we aim to tackle this real-world problem using two Memetic approaches that combine Genetic Algorithms with a Randomized Hill Climbing Local Search. The main goal of this thesis is to develop a metaheuristic technique to provide fast and close to accurate solutions to this problem, that can be applied practically in real-world situations. We aim to develop a Genetic Algorithm and a Local Search Algorithm, as well as develop two Memetic Algorithms by combining these two algorithms. Another aim of this thesis is to develop new problem-specific crossover and mutation operations to be used in the Genetic and Memetic Algorithms. We also aim to solve smaller instances with known exact solutions to optimality within practical, short running time. Additionally, we aim to solve larger instances, where a true optimal solution cannot be guaranteed, to feasibility with close to accurate results .

These aims are summarized as follows:

- Investigate metaheuristic solutions by developing a Genetic Algorithm and Local Search Procedure.
- Combine the Local Search Procedure with the Genetic Algorithm to create two Memetic Algorithms that can be practically applied in a real-world setting.
- Provide optimal solutions to small instance within practical running times.
- Provide close to accurate solutions to large instances within practical running times.
- Design new crossover and mutation operations to use in both the Genetic Algorithm as well as the Memetic Algorithms.

1.3 Contributions of This Thesis

Smaller instances of this problem variant have been previously solved by [5] using **Constraint Programming (CP)** techniques, yet there have been no solutions provided for larger instances with more than 60 packages and the problem has not been solved using a metaheuristic approach. In this thesis, we begin by providing a formal mathematical formulation of the problem. We then develop a metaheuristic algorithm to solve large instances of this crowdsourced **Vehicle Routing Problem (VRP)**. Namely a **Memetic Algorithm (MA)** where a **Genetic Algorithm (GA)** combined with a **Local Search (LS)** is developed. Two techniques of hybridizing **GA** and **LS** along with new crossover and mutation operations are developed. A Randomized Hill Climbing Algorithm is developed as our Local Search procedure using the mutation operation to define the neighborhood. These techniques and operations are experimented with, and the results are evaluated empirically. The resulting algorithms give feasible solutions within reasonable running times, and several large instances are solved that have been previously too large to solve. Instances with up to 135 packages are solved within running times less than 8 minutes.

During the development process, operations and methods are tested with unit tests and some results are checked using a solution validation process developed by [5]. The results of the two Memetic Algorithms as well as the results of the Genetic Algorithm on its own are evaluated on three sets of instances. The instance sets vary in size. The smallest set of instances is solved to optimality using Constraint Programming techniques; hence, the exact solutions are known. This set of instances is then used to evaluate the algorithms' ability to find the true optimal solution on these small instances. The two other instance sets have larger instances where the exact solution is unknown. On these instances, we compare the algorithms' solutions to some approximate solutions found using Constraint Programming techniques, and we compare the algorithms against each other when no exact or approximate solution is known. The feasibility of the results and running times of the algorithms on these larger instances are also evaluated.

The thesis presents these contributions:

- A Genetic Algorithm with two variations on a new problem-specific crossover is explored. The GA is evaluated and compared against Constraint Programming results.
- A problem-specific mutation operation is developed using a element swap approach.
- A Randomized Hill Climbing Local Search procedure is implemented with a neighborhood defined using the mutation operation.
- By combining the Hill Climbing procedure with the Genetic Algorithm, two Memetic Algorithms are experimented with and compared to Constraint Programming results. The two Memetic algorithms can solve large instances with up to 135 packages.
- Before the empirical evaluations, the Genetic and Memetic Algorithms' parameters are experimentally tuned yield feasible solutions.
- Using the Genetic Algorithm and the Memetic Algorithms, feasible solutions to small instances with up to 14 packages are obtained with average running times of less than 0.4 seconds.
- Feasible solutions better than CP results are found using the Memetic Algorithms with running times less than 6 seconds on instances with 33 to 48 packages.
- Feasible solutions to large instances with up to 135 packages are obtained in less than 8 minutes that are too large to solve using exact methods within two hours.

1.4 Structure Of This Thesis

In Chapter 2 we provide a formal definition of the problem and review the recent work on similar problems in the current literature. After providing a mathematical definition of the problem, recent work on similar problems is examined. The exploration of the related

work begins with an investigation of capacity and distance constrained VRP with open ended routes (Open VRP). Such problems have many similarities to ours. Next, [VRP with occasional drivers \(VRPOD\)](#) first introduced by [\[8\]](#) in 2016 is discussed. Finally, we examine similar Memetic Algorithms on Open VRPs. In Chapter [3](#), we begin by discussing the techniques used in the implementation of the Genetic Algorithm. In this section, section [3.1](#), we provide a background of Genetic Algorithms and aspects to consider when developing such an algorithm. The fitness function is also discussed in this section. Furthermore, we define the solution representation, and how the crossover and mutation operations. In the next section of Chapter [3](#), section [3.2](#), the Memetic Algorithms are discussed. This begins with a background on Memetic Algorithms and then explores how the Genetic Algorithm is combined with a Local Search procedure to create our two main Memetic Algorithms. In the final section of Chapter [3](#), section [3.3](#), we discuss the design of the Local Search procedure. Chapter [4](#) presents the experiments performed on several sets of problem instances and the meaningful findings of the experiments are discussed. Finally, we conclude our findings in Chapter [5](#). Here our work is summarized, and the overall results of the experiments are reviewed. In addition, possible future work that could be done on Memetic approaches to this problem and further problem variants that could be explored is discussed in Chapter [5](#).

Problem Statement and Related Work

The traditional approach to package delivery is analogous to the classical Vehicle Routing Problem first introduced in 1959 [9]. The Vehicle Routing Problem is an optimization problem where vehicles are routed to service a number of customers and the routes are optimized in some way. Most often **VRP** is defined to include a central depot location where all vehicles leave from and return to. There are many variants of the Vehicle Routing Problem; common variations include capacity constraints [10, 11], distance constraints [10], open-ended routes [12] where drivers do not return to the central depot after their route, and many more variations and constraints.

Several variants of this general problem have been tackled in the past, but this thesis will focus on a new, real-world variant that appears in the industry. Here, the problem is a crowdsourced package delivery problem where the goal is to deliver packages from a single depot to customers in various locations. Delivery drivers are crowdsourced. The drivers are compensated by the number of packages they deliver and other factors are not considered in driver compensation. The goal then, is to provide these drivers with reasonable routes and to minimize the total distance that all the drivers deviate from their original route. Minimizing the total deviations of all the vehicles will in turn reduce fuel costs and emissions.

In this chapter, we present the formal definition of the problem, explore similar problems, and explore the current state-of-the-art approaches to these similar problems.

2.1 Problem Statement

The problem is a Constraint Optimization problem and a Vehicle Routing Problem. A set of packages, P , and a set of drivers, A , are given. For each package $p \in P$, a destination,

$dest_p$, and a volume, v_p , are provided. For each driver $a \in A$, three attributes are provided: the drivers' original destination $dest_a$, the maximum volume their vehicle accommodates v'_a , and the maximum distance they are willing to deviate from their original path $maxDev_a$. Also given is the maximum number of packages a vehicle can deliver, $maxP$. The value $maxP$ is the same for every vehicle.

The goal is to provide routes to these drivers such that the sum of the deviations over all the drivers is minimized, while simultaneously fulfilling several constraints. A **route**, r_a , for a vehicle $a \in A$ is an ordered sequence beginning with the first packages to be delivered by vehicle a and ending with the last package to be delivered by vehicle a . A **solution** would then be a set of these routes, one for every driver in A . The deviation of driver a , dev_a , is defined as the difference between the distance traveled by vehicle a along route r_a and vehicle a 's original route. Several constraints must be satisfied by these routes. These constraints are hard constraints, meaning a solution is not feasible unless all the constraints are satisfied.

- **Package Delivery Constraint** Each package must be delivered and a package must not be assigned to multiple cars (i.e. each package must be assigned to *exactly* one car).
- **Volume Constraint:** The maximum volume each car can accommodate, v_a , must not be exceeded.
- **Number of Packages Constraint:** The maximum number of packages a driver can deliver, $maxP$, must not be exceeded by the route provided to any vehicle.
- **Deviation Constraint:** The maximum distance each driver is willing to deviate, $maxDev_a$, must not be exceeded by the route provided for the vehicle.

The **objective** of this constraint problem is to minimize the sum of the deviation of all the vehicles, $\sum_{a \in A} dev_a$, while also satisfying all of the above constraints.

2.1.1 Formal Problem Definition

We can formulate this problem formally as follows.

Constants:

$P = \{1, 2, \dots, n\}$ represents the packages to be delivered.

$A = \{1, 2, \dots, m\}$ represents the vehicles available to make deliveries.

$v_i :=$ the volume of package i , $\forall i \in P$.

$v'_i :=$ the volume of vehicle i , $\forall i \in A$.

$s_i :=$ the distance from the depot to the destination of package i , $\forall i \in P$.

$s'_i :=$ the distance from the depot to the destination of vehicle i , $\forall i \in A$.

$d_i^j :=$ the distance from package $i \in P$ to package $j \in P$.

$f_i^j :=$ the distance from the destination of package i to destination of vehicle j , $\forall i \in P$ and $\forall j \in A$.

$maxP :=$ the maximum number of packages a vehicle can transport.

$maxDev_i :=$ the maximum distance a vehicle is willing to deviate from its original route.

Variables:

$l_i :=$ the number of packages delivered by vehicle i , $\forall i \in A$.

$r_i = \{x_i^1, x_i^2, \dots, x_i^{l_i}\} :=$ the ordered set of all packages delivered by vehicle i . This represents the route for vehicle i , $\forall i \in A$.

$x_i^k \in r_i :=$ the k th package delivered by vehicle i , $\forall i \in A$ and $\forall k \in \{1, 2, \dots, l_i\}$.

Objective:

$$\text{MIN} \sum_{i \in A} dev_i \quad (2.1)$$

$$\text{where } dev_i = s_{x_i^1} + \sum_{k < l_i - 1} (d_{x_i^k}^{x_i^{k+1}}) + f_{x_i^{l_i}}^i - s'_i$$

Subject to the following constraints:

$$\forall p \in P, \exists i \in A \text{ s.t. } p \in r_i \quad (2.2)$$

$$\forall p \in P \text{ if } \exists i \in A \text{ s.t. } p \in r_i \text{ then } p \notin r_j \quad \forall j \in A \setminus \{i\} \quad (2.3)$$

$$\sum_{p \in r_i} v_p < v'_i \quad \forall i \in A \quad (2.4)$$

$$l_i < maxP \quad \forall i \in A \quad (2.5)$$

$$dev_i < maxDev_i \quad \forall i \in A \quad (2.6)$$

Equations 2.2 and 2.3 represent the Package Delivery Constraint. Equation 2.2 ensures that each package is assigned to a vehicle and is delivered. Equation 2.3 ensures that no package is assigned to more than one vehicle. Equation 2.4 represents the Volume Constraint, ensuring that the total volume of the packages assigned to each vehicle does not exceed the vehicles maximum volume v'_i . Equation 2.5 ensures that the number of packages assigned to each vehicle does not exceed $maxP$; this satisfies the Number Of Packages Constraint. The final equation, Equation 2.6, ensures that every vehicle's route does not exceed the vehicle's allowed maximum deviation.

2.2 Related Work

Although this is a new problem, it shares many aspects with similar problems found in the literature. Our problem is a variant of the VRPOD where occasional drivers make up the entire fleet rather than supplement a standard fleet. Two capacity constraints are included in our problem: a volume constraint that is heterogeneous across the fleet and a homogeneous constraint on the discrete number of packages a vehicle can deliver. Deviation constraints are also included and instances can be such that occasional drivers deliver more than a single package. In this problem, the routes can be considered similar to "open" routes in Open Vehicle Routing Problem since the drivers do not return to the depot after their deliveries. However, unlike the Open Vehicle Routing Problem, where drivers end on their final delivery, the drivers go to their own final destination after their final delivery, and that distance must be considered in the calculations of distance and deviation.

After an extensive review of the literature on the Vehicle Routing Problem, we found a large amount of research on several problems that share similarities to our proposed problem. In this section, we will first discuss similar and interesting algorithms for the Open VRP (OVRP) with capacity and/or distance constraints. We will look at the current research on these problems since they have some similarities to our problem. Since our problem is a variant of VRPOD, we will discuss VRPOD and recently proposed algorithms that solve VRPOD variants effectively and efficiently. Lastly, we will look again at Open VRPs and recent work where Memetic approaches similar to our proposed algorithms are used.

2.2.1 Capacity and Distance Constrained Open VRP

In this section we will discuss recent and related work on capacity and distance constrained VRPs with open ended routs. Our problem is a VRPOD where the occasional drivers make up the entire delivery fleet. However, our problem has several other constraints that are common in VRP variants. For example, the Open Vehicle Routing Problem (OVRP) - first introduced in 1997 [13] - is similar to our problem since the drivers do not return to the depot at the end of their deliveries. Our problem is also capacity constrained and could be considered distance constrained, albeit the distance is constrained by drivers *deviations* from their original path. It is still important to look at work previously done

on standard distance constrained problems. There has been a significant amount of work done on [VRP](#) in general [\[10\]](#). We will address some recent and related work on capacity and distance constrained [VRPs](#) and Open [VRPs](#) since they have many similarities to our problem.

An Open Vehicle Routing Problem with capacity and distance constraints was solved by [\[14\]](#) using a biased random-key GA. In a random key GA, the chromosomes are a set of fractional values between 0 and 1. The chromosome cannot represent aspects of the solution directly and must be decoded into a solution [\[14\]](#). The authors use benchmark [VRP](#) instances to evaluate the algorithm, and solve distance and capacity constrained versions with open routes [\[14\]](#). They compare their solutions with previous [OVRP](#) solutions and obtain promising results [\[14\]](#). Their solutions to instances with around 100 vertices (customers and depots) are competitively better than previous solutions, and the algorithm can solve instances with up to 440 vertices [\[14\]](#).

Previous work on capacity constrained [OVRP](#) (but not distance constrained) include a hybrid evolutionary algorithm by Repoussis et al. [\[15\]](#). This algorithm is much like a classical [GA](#) but it uses only mutation on individuals and no recombination or crossover is used on individuals [\[15\]](#). However, for each individual vector, there is also a "strategy parameters" vector where each element represents the probability that the corresponding element in the individual's chromosome will be mutated [\[15\]](#). These "strategy parameter" vectors evolve using a recombination operation rather than a mutation like their solution representation counterpart [\[15\]](#). The authors combine this evolutionary algorithm with the common metaheuristic Tabu Search [\[15\]](#). Every offspring from the mutation is further improved using the Tabu Search [\[15\]](#). The results of this algorithm are compared to previous solutions on [OVRP](#) instances and are shown to be competitive with previous work [\[15\]](#).

2.2.2 VRP with Occasional Drivers

Since our problem is a [VRPOD](#) variant, in this section we explore related work on several variants of the [VRPOD](#). In 2016 the Vehicle Routing Problem with Occasional Drivers ([VRPOD](#)) was introduced by Archetti et al. [\[8\]](#). The [VRP](#) variant discussed in [\[8\]](#) uses crowdsourced ad-hoc drivers to supplement a standard fleet of delivery trucks. [\[8\]](#) evaluates the cost-effectiveness and benefits of using occasional drivers and considers the various schemes for compensating the occasional drivers. The authors propose an Integer Programming formulation of the new problem, which they solve using CPLEX. CPLEX is commonly used Mathematic Programming solver by IBM [\[16\]](#). A Multi-Start heuristic is also employed by the authors in [\[8\]](#) to solve the [VRPOD](#) and the results are compared against the Integer Programming solutions. Since this introduction of the [VRPOD](#), many variants have been solved using numerous techniques both approximate and exact [\[17, 18, 8, 19, 20, 21, 22, 23, 24, 25\]](#).

In 2021, [\[18\]](#) uses a new "Evolutionary Multitasking" algorithm to simultaneously solve multiple instances of a [VRPOD](#) variant. The variant of the problem considered in [\[18\]](#)

has heterogeneous vehicle capacities and time window constraints. In this problem, the occasional drivers again supplement a regular fleet of delivery vehicles [18]. The Evolutionary Multitasking algorithm takes multiple instances of the problem as input, and the chromosomes can encode a solution to any of these instances [18]. Their algorithm is able to return a solution to each of the instances. This multitasking algorithm showed quality results in terms of both solution accuracy and efficiency [18].

Fabian Torres et al., [18], investigate a new variant of VRPOD where "stochastic destinations" are used. In the article, the term "stochastic destinations" means the occasional drivers have random destinations that may vary over separate days [19]. Their problem is titled the "Open VRP with Stochastic Destinations" [18]. In this article, the problem is formulated as a Mixed Integer Program (MIP) and is solved using a branch-and-price approach. Their approach was able to solve instances where up to 50 customers are served by the vehicles [18]. The authors improve on this result by employing a Column Generation Heuristic to relax the MIP formulation. With this heuristic they can solve instances with up to 100 customers [18].

In an article in 2019, Macrina et al., [17], develop a Variable Neighborhood Search Heuristic for a variant of the VRPOD that includes time windows and "transshipment nodes" [17]. In their problem, smaller warehouses, called "transshipment nodes", are used throughout the city or delivery area [17]. At these "transshipment" warehouses, the occasional drivers make pickups for their deliveries and the "transshipment" warehouses are stocked using a standard fleet trucks [17]. Macrina et al. [17] show the cost advantages of both the use of occasional drivers as well as of the use of these "transshipment nodes". They formulate the problem as a Mixed Integer Program and solve it using CPLEX, and develop a Variable Neighborhood Search Heuristic (VNS) for the problem. The VNS is efficient and effective compared to the CPLEX results and is capable of solving instances with 50 customers [17].

In 2020, the same problem variant that was studied by [17] is also solved by [20] using Greedy Randomized Adaptive Search Procedures (GRASP) [20]. The results of this approach are compared to previous results on the problem. The GRASP approach is both efficient and effective in providing solutions to the problem [20].

In a 2019 paper, Abu Al Hla et al. [25] develop a complex VRP problem using occasional drivers [25]. In an interesting combination of constraint optimization and behavioral economics, the authors develop an MIP formulation where driver behavior is included. This is done by allowing drivers to choose routes, and driver behavior is modeled using a parameter for risk adverseness [25]. In this article [25], the problem is solved to near optimality with the use of two different heuristics, a *Greedy Heuristic* and an *Intra-Route Heuristic Neighborhood Search*.

Sampaio et al. [24] use an Adaptive Large Neighborhood Search metaheuristic to solve a variant of VRPOD [24]. Their variant includes a time window constraint and transfers [24]. Transfers means a driver can take the package part of the way to a location where another driver will pick it up from that location and complete the delivery [24].

2.2.3 Memetic and Genetic Algorithms on OVRP

In this thesis, we develop a Memetic Algorithm to solve our crowdsourced Vehicle Routing Problem. To the best of our knowledge, there has been little to no research using Memetic Algorithms where [GA](#) is combined with a Local Search to solve a VRPOD variant. However, there is promising research on other [VRP](#) variants using Memetic Algorithms. As discussed before, the [OVRP](#) is similar to our problem, especially when capacity and deviation constraints are included in the [OVRP](#). In this section, we will look at recent research using Memetic Algorithms and Hybrid Genetic Algorithms on problems with open routes. Over the years there have been several Memetic Algorithms proposed on various OVRP variants: [\[26, 27, 28, 29\]](#).

Ran Liu et al. [\[28\]](#) propose a [Hybrid Genetic Algorithm \(HGA\)](#) where a Local Search is used instead of a mutation operation. Every child chromosome from the recombination process has a chance that the Local Search methods will be applied to it [\[28\]](#). Their algorithm is used to solve a multi-depot variant of the OVRP. The authors also formulate their problem as an [MIP](#) and solve it using CPLEX. The results of their [HGA](#) are compared to both CPLEX results and results from another metaheuristic algorithm called a “list based threshold accepting heuristic” [\[28\]](#). The [HGA](#) algorithm outperforms both CPLEX and the other heuristic, especially on moderate to large sized instances with 100 to 350 customers [\[28\]](#). These are impressive results [\[28\]](#).

Richard Y.K. Fung et al. [\[26\]](#) explore an [Open Capacity Constrained Arc Routing Problem \(OCARP\)](#). In Arc Routing Problems, instead of all locations (nodes in a graph) needing to be visited by the vehicles, all arcs in the graph must be visited. The authors propose a Memetic Algorithm similar to the one proposed for the multi-depot OVRP by Ran Liu et al. [\[28\]](#). In the algorithm by Fung et al. [\[26\]](#), at every generation, each new child has a chance of being improved upon using a Local Search. The authors use three different operations within their Local Search to achieve a better solution. The results of this algorithm on random instances outperformed CPLEX on a lower bound Linear Programming formulation of the [OCARP](#) problem [\[26\]](#). They also compare the results of the Memetic Algorithm to that of a classical [GA](#) and the Memetic Algorithm outperforms the [GA](#) in all cases [\[26\]](#). The Memetic Algorithm was also compared to previously proposed heuristics for Capacity Constrained Arc Routing Problems (CARP) on commonly used benchmark instances [\[26\]](#). The Memetic Algorithm was competitive with the other heuristics [\[26\]](#). The [OCARP](#) problem discussed was quite different from ours since it is an Arc Routing Problem. An Arc Routing Problem is not the same as a Vehicle Routing Problem. Nevertheless, these findings are incredibly interesting results on the use of a Memetic Algorithm to solve a routing problem.

In an article published in 2021, Liang Sun et al., [\[27\]](#), employ a Genetic Algorithm for a specific variant of [OVRP](#) where time windows are considered, and travel times are uncertain. Because of the uncertainty, the authors propose a “light-robust-optimization model” [\[27\]](#). They develop a Genetic Algorithm with a self-adaptive mutation rate that increases if the *quality* of the population is low [\[27\]](#). The *quality* of the population is

2. PROBLEM STATEMENT AND RELATED WORK

determined by the fitness of the best solution in the population [27]. The authors also have a self-adaptive crossover rate that is decreased for individuals with a problem-specific attribute that leads to poor solutions. They use a “hybrid tournament selection” where the winners of a tournament are determined by not only the fitness of the chromosome but also by an attribute that tends to lead to poor solutions [27].

The Algorithms

This chapter introduces a Genetic Algorithm and a Local Search Algorithm, which are then combined to create two Memetic Algorithms. We begin with the Genetic Algorithm development. Some background on Genetic Algorithms is provided as well as some aspects to be considered when developing a Genetic Algorithm. These aspects include a solution representation, a fitness function, a crossover and mutation operation, and selection and replacement strategies. In the Genetic Algorithm section of this chapter, we discuss the solution representation that is developed in this thesis and the fitness function. These methods are used not only in the Genetic Algorithm but also in the Local Search and the Memetic Algorithms. We then discuss the design of our new crossover and mutation operations. Lastly the selection and replacement strategies used in the Genetic Algorithm are discussed. In the next section of this chapter, beginning with a brief background on Memetic Algorithms, we explore the development process of our Memetic Algorithms. In the final section of this chapter, we address the Local Search algorithm. As our Local Search procedure, we develop a Randomized Hill Climbing procedure with a neighborhood defined using the mutation operation. We also discuss in detail how the Local Search is integrated into the Memetic Algorithms.

3.1 Genetic Algorithm

A Genetic Algorithm is a metaheuristic algorithm that is inspired by and mimics the evolutionary process and natural selection [6]. In a Genetic Algorithm, a **chromosome** is generally a list of numbers or digits (**elements**) that represents a solution to the instance being solved [6]. A chromosome is also sometimes be referred to as an **individual** or a **solution** [6]. At the initialization of a GA, a set of chromosomes, called the **population**, is generated often randomly [30]. At every iteration of a Genetic Algorithm, called a **generation**, there is a new set of chromosomes that make up the population [30]. For every generation, several individual chromosomes are selected based on some selection

strategy [30]. These selected individuals are called **parents** [30]. In the next step, each selected pair of parents has a chance to be recombined using the **crossover** operation to create new **children** chromosomes [30]. If the parents are not crossed-over, the parents themselves are used as their **children** in the next generation [30]. After the crossover, each child has a chance of being **mutated** with the mutation operation [30]. A mutation operation uses only one parent chromosome and it involves a simple operation such as flipping a bit (when the chromosome elements are 1 or 0) or swapping two elements within the chromosome [6]. After the crossover and mutation phase, the newly created children replace most or all the individuals in the population [30, 6]. Hence a new population is created for the next generation. This process continues until some stopping criterion is met, for example, a maximum number of generations without improvement on the best fitness in the population [6]. Once a stopping criteria is met, the individual with the best fitness is returned.

Several things are required to develop a Genetic Algorithm. A crossover operation, a mutation operation, a solution representation that will be used as a chromosome, a selection, and a replacement strategy are required. In this thesis, a solution representation is developed and is used in both the Genetic Algorithm and the Local Search algorithm. A crossover operation and a mutation operation are developed for the Genetic Algorithm. The mutation operation is also used in the Local Search algorithm to define the neighborhood. A **Roulette Wheel** selection technique is used, where the chance of an individual being selected is proportional to that individual's fitness value. A replacement strategy where the children replace the chromosomes that they are most similar to is used.

3.1.1 Solution Representation

The first step in the algorithm development process is to define a solution representation to be used both as a chromosome in the Genetic Algorithm as well as used as an individual in the Local Search process. The solution representation must be well-suited for developing **GA** operations and must work well in our Local Search. The solution representation must also fully represent a solution to a problem instance. Therefore, it must express which packages are to be delivered by which vehicles, as well as the order in which these packages are to be delivered. It is vital in a real-world situation that our algorithms output a solution to the instance. Hence, the solution representation must be adaptable such that the algorithms can output solutions where the drivers are given the specific packages they must deliver and their delivery routes. In order to provide such routes to the drivers we must also represent the order in which packages are delivered.

We develop the following as our solution representation: The packages and the vehicles are both represented as an integer value starting at index 1. An array of length equal to the number of packages is used with the index representing a package and the array elements representing the vehicle assigned to deliver that package. A vehicle can appear several times in the array but no more than the number of positions available in each vehicle. [3, 2, 3, 1] is an example of such a solution representation for an instance with 4

packages, 3 vehicles and 2 positions per vehicle. In this example, packages 1 and 3 are delivered by vehicle 3, package 2 is delivered by vehicle 2, and package 4 is delivered by vehicle 1.

This solution representation functions well as a chromosome in the Genetic Algorithm and leads to the development of some interesting and effective crossover and mutation operations. The representation also works well in the Local Search, combining it with the mutation operation we are able to define a neighborhood in the Local Search procedure. Another advantage of this representation is that due to the length of the array being equal to the number of packages, the constraint that all packages must be delivered is satisfied for all solutions that are represented by this chromosome.

As previously noted, it is required that the order in which the packages are delivered must be provided by the algorithm in order for routes to be given to the drivers. However, including this in our solution representation and allowing the genetic operations to change the order of delivery would add many solutions to the search space that have poor fitness and fail to guide the algorithm toward better solutions. This would be detrimental to the algorithms' performance since many unimportant and similar solutions would be explored needlessly. However, it is still vital that the order of delivery be taken into account when an individual is evaluated. We always consider the best order for those packages to be delivered in the calculation of solution fitness. To achieve this we need to find the best order of delivery for each vehicle in each solution, this is done with brute force, by iterating over all possible orders of delivery to find the optimal order. Since instances do not usually have a large value for $maxP$ (the number of packages a vehicle is able to deliver) this brute force calculation is not excessively computationally expensive.

3.1.2 Initialization

To begin the Genetic Algorithm an initial population must be generated. We initialize the population by creating a large number of random solutions. The size of the population is controlled by a parameter we call p . This parameter is a percentage of the length of the chromosomes. Hence the population size is equivalent to $p \times P$ where p is the population size parameter and P is the number of packages in the instance.

To generate random individuals for each package we choose a random available vehicle and assign it to deliver that package. If that assignment violates the Volume Constraint, then a new vehicle is chosen. A vehicle becomes unavailable when it has been assigned to deliver $maxP$ packages. This ensures that the Number of Packages Constraint is not violated. Due to the length of the solution representation, the Package Delivery Constraint is also satisfied by every random solution. Hence, when the chromosomes are initialized they do not violate the Volume Constraint, the Number of Packages Constraint, or the Package Delivery Constraint

3.1.3 Fitness Function

To determine which solutions are better than others we need to define a fitness function to calculate a fitness value for each individual. The fitness function will be used in both the Genetic Algorithm and the Local Search Algorithm, which seek to minimize this fitness value. Within our fitness function, we include a penalty value that is added when the Deviation Constraint is violated. We define our fitness function to be the sum of the deviations of all the vehicles plus this penalty value. In our problem, a solution is considered a good solution if it satisfies all the constraints outlined in the problem statement while also minimizing total deviation over all vehicles. We call a solution **feasible** if all the constraints are satisfied: the Package Delivery Constraint, the Volume Constraint, the Number of Packages Constraint, and the Deviation Constraint. The first three constraints (i.e. the Package Delivery Constraint, the Volume Constraint, and the Number of Packages Constraint) are satisfied by the solution representation and remain satisfied after each crossover or mutation operation. The Deviation Constraint is not so easily satisfied since there are too many combinations to efficiently calculate the deviations of each vehicle. Because of this, the Deviation Constraint is implemented as a penalty within the fitness function. The fitness function calculates the fitness of an individual by adding the combined total deviations of all the vehicles (i.e. the problem's objective) to a penalty value that is added when the solution violates the Deviation Constraint. Since the Deviation Constraint is a hard constraint the solution is still considered infeasible when the constraint is violated. Hence, it is important to ensure our algorithms consider feasible solutions better than infeasible solutions. To do this, we need to ensure that every feasible solution has a smaller fitness value than any solution where the deviation constraint is violated. We then define the fitness function as follows:

Let A be the set of all vehicles, dev_i the deviation of vehicle $i \in A$, $maxDev_i$ be the maximum allowed deviation for vehicle $i \in A$, and A' be the set of cars that exceed their maximum deviation allowance: $A' = \{x \text{ s.t. } x \in A \text{ and } dev_x > dev_x^{max}\}$. The fitness function is defined as

$$\text{fitness} = \sum_{i \in A} dev_i + m \sum_{j \in A'} (dev_j - maxDev_j) \quad (3.1)$$

where the penalty m is defined as

$$m = \sum_{i \in A} maxDev_i \quad (3.2)$$

The worst possible fitness for a feasible solution is when all vehicles meet their allowed deviations exactly: $\sum_{i \in A} maxDev_i$. Hence, by defining m such that $m = \sum_{i \in A} maxDev_i$, Function 3.1 ensures that any individual that violated the Deviation Constraint would have a fitness greater, by a large amount, than that of even the worst possible feasible individual. The penalty also takes into account the amount a vehicle exceeds its maximum

deviation allowance. Therefore, a solution where the vehicles only slightly exceed their maximum allowance is considered better than a solution where the deviation allowances are exceeded by a large amount. This helps guide the Genetic and Local Search algorithms toward better solutions and feasibility.

3.1.4 Crossover

A significant aspect to be considered when designing a Genetic Algorithm is the crossover operation. A crossover operation generally takes two chromosomes, selected from the population during the selection process, and recombines them with each other in some way producing two new chromosomes. Input chromosomes are called **parents** and the chromosomes produced by the operation are called **children**. When designing the crossover operation, there are some requirements we want the operation to satisfy. The first requirement is the crossover must guarantee the satisfaction of the first three problem constraints - the Deviation Constraint is not included since it is handled in the fitness function. This means that given two parents satisfying these constraints, the resulting children must also satisfy the same constraints. The second requirement is that the newly created children should take important aspects from both parents. We develop a new crossover operation and experiment with two different variations on that operation. The resulting crossover both guarantees that the children do not violate the constraints, and allows the children to take elements from both parent chromosomes.

The crossover operation is defined as follows: Given two parent solutions, beginning with the first index the vehicle at that index in the first parent is exchanged with the vehicle at that index in the second parent and vice versa. This exchange is only made if such an exchange of vehicles does not violate any of the three non-penalty constraints (the Package Delivery Constraint, the Volume Constraint, or the Number of Packages Constraint). If an exchange violates one of these constraints we say the exchange is “illegal”. We then move to the second index and exchange those two vehicles if it is a legal exchange. This process continues until the end of the chromosome is reached or a predetermined crossover point is reached. The resulting children are later used in the next generation of the [GA](#) population. [Figure 3.1](#) shows the first steps in the process of element exchanges between two parent chromosomes. In this example, we assume $maxP = 2$. We see at the 2nd step, that the exchange is illegal since making this exchange would result in vehicle 2 holding 3 packages; therefore the exchange is not made. [Figure 3.2](#) shows the final result of the crossover operation when exchanges are made until the end of the chromosomes.

This crossover operation fulfills the requirement that all three of the non-penalty constraints remain satisfied. The Package Delivery Constraint, that all the packages must be delivered, is trivially satisfied by the solution representation since the length of the chromosomes remain unchanged by the crossover operation. The next two constraints, the Volume Constraint and the Number of Positions Constraint were more computationally expensive to satisfy. These constraints must be checked at every vehicle exchange, but in the end, the constraints remain satisfied since an exchange is not made if one of the

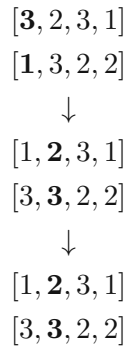


Figure 3.1: Crossover operation: first steps

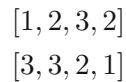


Figure 3.2: Crossover operation: final result

constraints is violated by the exchange. At every step in the crossover operation, the chromosomes do not violate the constraints, so the final results also do not violate the constraints. These checks at every index contribute to most of the complexity of the crossover function.

We experiment with two variations on this crossover. In the first variation, we iterate through the entire chromosome when making the exchanges. We will refer to this variation as **crossover-1**. In the second variation, exchanges are only made until the midpoint of the chromosome is reached, this we use as a crossover point. We refer to this variation as **crossover-2**.

3.1.5 Mutation

Another operation needed in a Genetic Algorithm is a mutation operation. We develop a simple chromosome element swap operation, where two vehicles swap places in the solution representation. Below is an example of such an operation:

$$[3, 2, 3, 1] \xrightarrow{\text{mutate}} [3, 1, 3, 2]$$

This operation represents the real-world action of swapping two packages between two vehicles. In our mutation operation, the two elements that are to be swapped are chosen

at random from the solution representation. If a swap violates the Volume Constraint or the Number of Packages Constraint the swap is not made and two new elements are randomly chosen to swap. In this way, the operation is able to ensure that the mutated solution satisfies the non-penalty constraints, given that the constraints are satisfied before the mutation operation is applied. The mutation operation also guarantees the satisfaction of the Package Delivery Constraint since the operation does not change the length of the solution representation array.

This mutation operation is used on the new children after the crossover operation. There is a small chance for each child to be mutated using the operation. This chance of mutation is controlled by the mutation rate parameter μ . The mutation operation is also used to define the neighborhood of the Randomized Hill Climbing algorithm. We define an immediate neighbor as an individual that can be reached with one mutation operation.

A slight drawback to our mutation operation is that it does not allow the possibility of vehicles with no packages assigned to them in the original chromosome to gain packages in the mutated chromosome. Because of this, it is important that these unused vehicles can be incorporated into new individuals that are created from the crossover operation. The new vehicles will come from the other parent individual during the crossover process. This makes diversity vital to our algorithms' population. A possible fix for this problem is to use a different mutation operation, for example, a *flip* operation where a single element is changed to any other available vehicle. This however presents its own drawbacks. Another option would be to somehow combine our mutation with another operation that allows for the infusion of unused vehicles, like the method mentioned above. We did not develop such an operation due to concerns about the computation time of such a mutation. We also believe this issue is not significant, the crossover operation combined with a diverse population should be sufficient to counteract any negative effects this issue may cause. However, this is an interesting area that could be explored in further research.

3.1.6 Selection

In the selection process of a Genetic Algorithm, individuals in the population are chosen to recombine and create the next generation of the population. In the selection process, two solutions are chosen from the current population, they are then recombined using the crossover operations to produce two new children solutions. Then two new solutions are chosen from the population and the process is repeated until some or all of the population has been selected and reproduced. The number of children created during this process is controlled by a parameter called σ . The value of this parameter is a percentage of the population size. The selection process continues until the number of children created equal to the product of the size of the population and the parameter σ . The children are inserted into the population for the next generation using a replacement strategy.

The parents are selected for reproduction using a Roulette Wheel Selection. In Roulette

Wheel Selection each individual in the population has a chance of being selected based on its fitness value. Two parents are then chosen from the population based on these chances. The pseudo-code in Algorithm 3.1 shows the steps of the selection process. In the pseudo-code, Algorithm 3.1, *solution.Fitness()* returns the fitness of that solution and *array.binarySearch(v)* does a binary search in the array for the value *v*, returning the position of the value just below *v* if *v* is not the array. This finds the position corresponding to the solution that the random value *v* will select.

Algorithm 3.1: Roulette Wheel Selection

Input : *population, σ, χ, μ*
Output : *children*

```
1 children ← new empty array
2 for  $i < (population.size * selection.Size)/2$  do
3   wheel[0] ←  $1/(population[0].Fitness())$ 
4   for  $j < population.size - 1$  do
5     | wheel[j + 1] ← wheel[j] +  $1/(population[j + 1].Fitness())$ 
6   end
7   randomValue1, ← random double between 0 and
   wheel[population.size - 1]
8   randomValue2, ← random double between 0 and
   wheel[population.size - 1]
9   index1 ← wheel.binarySearch(randomValue1)
10  index2 ← wheel.binarySearch(randomValue2)
11  parent1 ← population[index1]
12  parent2 ← population[index2]
13  if random double between 0 and 1 <  $\chi$  then
14    | child1, child2 ← Crossover(parent1, parent2)
15  end
16  else
17    | child1 ← parent1 child2 ← parent2
18  end
19  if random double between 0 and 1 <  $\mu$  then
20    | child1 ← Mutate(child1)
21  end
22  if random double between 0 and 1 <  $\mu$  then
23    | child2 ← Mutate(child2)
24  end
25  children.add(child1) children.add(child2)
26 end
```

Once two parents are selected they are either crossed-over to create two new children or they remain unchanged and are used as the resulting children. The chance that a pair of parents are crossed-over is controlled using a parameter χ . At this point, there is also a

chance that the resulting children are mutated. This is also controlled with a parameter μ

3.1.7 Replacement

In the next step of the Genetic Algorithm, the newly created children replace all or some of the individuals in the current population to create the next generation of the population. This is the replacement process. There are several techniques for replacement that are often used in Genetic Algorithms.

In our replacement strategy, we use an elitist strategy where the best individual in the population is never replaced. All the children created are inserted into the population replacing the individuals in the old population that are most similar to them, unless the individual to be replaced is the best individual in the current population. The number of individuals that are replaced is then roughly the same as the number of individuals that are selected during the selection process. If only some of the population is selected for reproduction then we only replace some of the population with the newly created children, keeping both some of old individuals that are dissimilar to the new children and the best individual in the population. This helps maintain a degree of diversity within the population. What portion of the population is selected for reproduction is controlled by the parameter σ . If $\sigma = 1$, then all of the population is selected- In this case, the only solution that is kept for the next generation is the elite solution. After all of the selected parents have been recombined, each of the children are then inserted into the population. The section of the population that is replaced is not necessarily the same section of the population that is chosen for reproduction.

This replacement is done using a strategy that replaces the solution that is most “similar” to each child. To use this strategy a measure of similarity or “closeness” between two chromosomes is needed. We use **hamming distance** as our distance measure. Hamming distance is defined as the number of index locations in the solution representation array where the element at that location differs. For example the chromosome [3, 1, 2, 2] would have a hamming distance of 2 from the chromosome [3, 3, 1, 2], since two of the chromosome elements differ.

This distance measure works well as a measure of similarity between two of the chromosomes. It does not take the actual numeric value of the chromosome’s elements into account, which is important because a vehicle’s value is arbitrary and carries no information about the vehicle or its destination. Several other commonly-used distance measures take the numeric value of the elements into account, so they were not considered for our distance measure. For example, Euclidean distance or Manhattan distance would not be suitable for our case. These distance measures would indicate that chromosome [3, 1, 2, 2] is closer to chromosome [3, 1, 3, 2] than to chromosome [1, 3, 9, 2], which would be detrimental as these element values do not hold this type of numeric or ordinal meaning. Another distance measure to consider would be to use our mutation operation as a way to measure the distance between individuals; for example, the number of swaps

needed to reach the other individual. This, however, has a major drawback in that it cannot consider unused vehicles. This is not an issue for the mutation operation because our crossover counteracted the effects but it is not suitable for the distance measure in our replacement strategy. Two solutions where one uses a vehicle and the other does not use that vehicle would be considered infinity distance from each other since they cannot reach each other using the mutation swaps. This would be detrimental if in every other aspect the two chromosomes are extremely similar or identical. For example, the chromosomes $[1, 2, 3, 4]$ and $[1, 2, 3, 5]$ illustrate this issue. These two chromosomes would be considered infinitely dissimilar, however we can see that they are similarly.

Algorithm 3.2 shows the pseudo-code for the replacement strategy. In the pseudo-code $HammingDistance(sol1, sol2)$ returns the hamming distance between the solutions $sol1$ and $sol2$ and $population.bestInPopulation()$ results the solution in the current population with the smallest fitness value.

Algorithm 3.2: Replacement

```
Input : children, population
Output : population
1 for child  $\in$  children do
2   mostSimilar  $\leftarrow$  population[0]
3   for i  $\in$  population do
4     if  $HammingDistance(i, child) < HammingDistance(mostSimilar, child)$ 
5       then
6         mostSimilar  $\leftarrow$  i
7     end
8   if mostSimilar  $\neq$  population.bestInPopulation() then
9     population.remove(mostSimilar)
10    population.add(child)
11  end
12 end
13 return population
```

3.2 Memetic Algorithm

A Memetic Algorithm is a Hybrid Genetic Algorithm where the evolutionary algorithms such as GA are combined with Local Search techniques [7]. Our aim, in taking a Memetic approach, is to capitalize on the best attributes of a Genetic Algorithm and a Local Search algorithm. A Genetic Algorithm can find promising areas of the search space while the Local Search Algorithm delves deeper into that specific area. There are many ways for these two methods to be combined, and many varying techniques were used in the literature: [31, 32, 33, 34, 35, 36, 37]. Often the Local Search is applied to some

individuals in the population at every generation of the GA.

We develop two Memetic Algorithms and refer to them as HA1 and HA2. Algorithms 3.3 and 3.4 as well as the flowcharts in Figures 3.3 and 3.4 show the process of each algorithm. In HA1 the Local Search is applied at each generation of the GA and is applied to the best solution in the current population. In HA2 the Local Search is applied at the end of the process and only to the final solution found by the GA. In both cases, a Randomized Hill Climbing Local Search is used, and the two crossover variants are can be used.

Algorithm 3.3: HA1

```

Input : Instance, stoppingCriteria, p,  $\mu$ ,  $\chi$ ,  $\sigma$ 
Output: Solution
1 population  $\leftarrow$  initialize population with  $p * Instance.P$ 
   individuals
2  $n \leftarrow 0$ 
3 currentBest  $\leftarrow$  individual in population with smallest Fitness
4 while  $n < stoppingCriteria$  do
5   | children[ ]  $\leftarrow$  SelectionCrossoverMutation(population,  $\sigma$ ,  $\chi$ ,  $\mu$ )
6   | population  $\leftarrow$  Replacement(population, children[ ])
7   | Solution  $\leftarrow$  individual in population with smallest Fitness
8   | Solution  $\leftarrow$  RandomizedHillClimbing(Solution)
9   | if Solution.Fitness  $<$  currentBest.Fitness then
10  |   |  $n \leftarrow 0$ 
11  |   | currentBest  $\leftarrow$  solution
12  |   | end
13  |   | else
14  |   |   |  $n \leftarrow n + 1$ 
15  |   |   | end
16 end
17 return Solution

```

3.3 Local Search

Over the years many approaches have been used in creating Memetic Algorithms and Hybrid Genetic Algorithms [11, 37]. We took two distinct approaches when combining a Genetic Algorithm with a Local Search. We used a Randomized Hill Climbing Algorithm as our Local Search procedure. The aim of the Local Search component is to explore a specific area of the search space while the GA component explores the space more globally. The Randomized Hill Climbing algorithm can only explore the immediate area near the current solution, and only take steps if the step shows improvement in fitness. Hence, if the algorithm is in a local valley, it will converge early on a local optimum and be unable to move out of the valley. However, for our purposes this is acceptable

Algorithm 3.4: HA2

Input : $Instance, stoppingCriteria, p, \mu, \chi, \sigma$
Output : $Solution$

```
1  $population \leftarrow$  initialize population with  $p * Instance.P$ 
   individuals
2  $n \leftarrow 0$ 
3  $currentBest \leftarrow$  individual in population with smallest  $Fitness$ 
4 while  $n < stoppingCriteria$  do
5    $children[] \leftarrow$  SelectionCrossoverMutation( $population, \sigma, \chi, \mu$ )
6    $population \leftarrow$  Replacement( $population, children[]$ )
7    $Solution \leftarrow$  individual in population with smallest  $Fitness$ 
8   if  $Solution.Fitness < currentBest.Fitness$  then
9      $n \leftarrow 0$ 
10     $currentBest \leftarrow solution$ 
11  end
12  else
13     $n \leftarrow n + 1$ 
14  end
15 end
16  $Solution \leftarrow$  RandomizedHillClimbing( $Solution$ )
17 return  $Solution$ 
```

behavior for the Local Search process of our algorithms. The detrimental effects of early convergence are to be counteracted by the Genetic Algorithms.

The same solution representation used in the [GA](#) is used for individuals in the Randomized Hill Climbing algorithm. We define the neighborhood as individuals reachable with one mutation operation. At every iteration of the Randomized Hill Climbing algorithm, a random mutation operation is applied to the current individual. This is equivalent to choosing a random neighbor from the neighborhood of the individual. If this neighbor has better fitness than the current individual then the neighbor becomes the new current individual. If the neighbor does not have better fitness than the current individual then no move is made and the current individual remains the same. This continues until no individual in the current neighborhood is better than the current individual or until a stopping criterion is met.

The Hill Climbing process is used at every generation in HA1. This greatly increases the running time of the algorithm. So in order to keep the running time reasonable, only a few iterations of Randomized Hill Climbing are used. A maximum of 10 iterations of Randomized Hill Climbing is used in HA1 along with a stopping criterion of 5 iterations without improvement on the current solution. Since, in HA2, the [LS](#) process is only applied to the final solution returned by the GA, HA2 excludes the possibility of a refined search of other areas in the search space, which HA1 is able to do. However, HA2 has

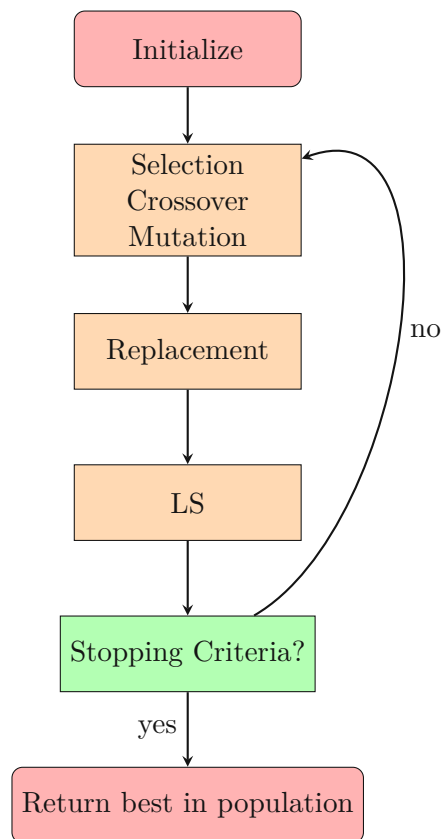


Figure 3.3: HA1

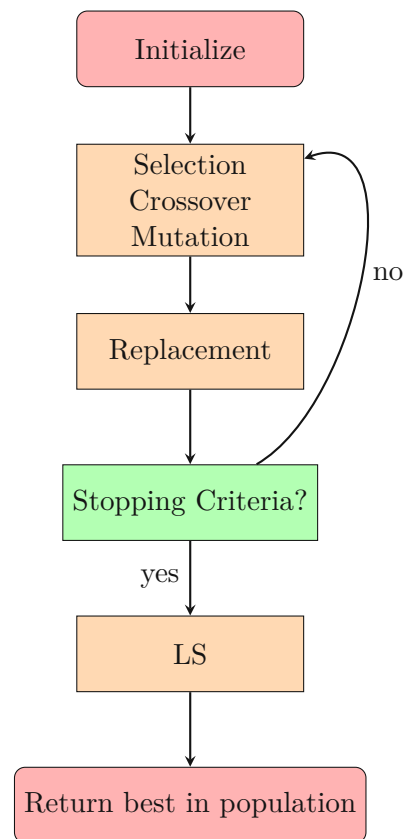


Figure 3.4: HA2

the benefit that many more iterations of the Hill Climbing procedure can be used since the **LS** procedure is only applied once. This allows HA2 to search the area in more detail and come closer to finding the local optimum at that point. For HA2, a maximum of 100 iterations is used, with a stopping criterion at 50 iterations without improvement.

Experiments and Results

In this chapter, the algorithms' parameters are tuned and experiments are performed on various sized instance. The Genetic and Memetic Algorithms results are compared to the results found by a Constraint Programming solver using the model proposed in [5]. We begin by randomly generating instances using an instance generator developed by [5]. Then instances are randomly selected, some are used for the parameter tuning process and the other randomly selected instances are used form the experiments. During the parameter tuning process, the parameters that yielded the most feasible solutions over all the instances are selected to be used in the experiments. In the experiments, the best fitness found by the algorithms are compared to each other and to solutions found by the Constraint Programming solver. All parameter tuning runs and experiments are done on a PC running Linux Mint 19.3 Cinnamon with the following specifications:

- CPU: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx
 - 4 Cores
 - 2.1 GHz
- RAM: 8GB
- GPU: Radeon 540X

For both the parameter tuning and the experiments, we generated three sets of random instances: small instance (5 to 15 packages), medium-sized instance (30 to 50 packages), and large instances (100 to 150 packages). We used a random instance generator previously implemented by [5] to generate instances. For each size range (small, medium, and large) we generated 20 instance.

Each instance includes P (the number of packages), A (the number of vehicles), $maxP$ (the maximum number of packages the vehicles are willing to deliver), the volumes

| Instance Attribute | Value Range |
|---|-------------|
| Package Volumes | 50-500 |
| Vehicle Available Volumes | 700-1500 |
| Maximum Auto Deviation | 10-30 |
| Max Number of Packages per Vehicle ($maxP$) | 1-4 |
| Package and Vehicle Destinations Coordinates | 25-40 |

Table 4.1: Instance Generation Value Ranges

available in each vehicle, and the maximum distances the vehicles are willing to deviate [5]. The coordinates of the package destinations and the vehicle destinations are randomly generated and are measured in kilometers, with the depot at the origin coordinate (0,0) [5]. This range is used by the instance generator to calculate the distances between each package destination, vehicle destination, and the depot [5]. These distances are then used in the instances [5], thus, each instance also includes the distances between the depot and the vehicles' destinations, the distances between the depot and the package destinations, the distances between each package destination, and the distances between the package destinations and the vehicles' destinations [5]. All distances are in kilometers and volumes are in cubic decimeters [5].

We used this instance generator to generate several sets of instances with various sizes. The number of packages is randomly generated from 5 to 15 for the small instances, 30 to 50 for the medium sized instances, and 100 to 150 for the large instances. The number of vehicles is proportional to the number of packages. For the other values we used the same ranges for all three sets of instances. $maxP$ is randomly generated between 1 and 4. The volumes available in each car is generated as a random number between 700 and 1500, and the package volumes are random values between 50 and 500. The maximum distances the vehicle drivers are willing to deviate are generated randomly between 10 and 30. The coordinates of the package and vehicle destinations are randomly generated between 25 and 40. These ranges are summarized in Table 4.1.

After generating 20 instances in each size range, we then randomly chose 7 from each set of 20. Next, using a CP implementation, previously formulated and implemented by [5] we solved these instances using the "Chuffed" CP solver in MiniZinc [38, 39] with a time limit of 2 hours. 2 instances from each set of 7 were randomly selected to be used in the parameter tuning process and the remaining 5 in each set are used in the algorithm comparisons.

4.1 Parameter Tuning

In order to evaluate the algorithms developed in this thesis, the parameter values must be selected to be used when running the algorithms for the experiments. Genetic Algorithms often have a large number of parameters. In our algorithms there are 4

parameters:

- p : the population size as a portion of P (the number of packages in the instance)
- μ : the mutation rate
- χ : the crossover rate
- σ : the selection size

After generating the random instances we randomly selected 2 instances from each set of 7 to use as the training instances to tune the parameters. Table 4.2 shows the instances used for the training data.

| Instance | Training Instances | | |
|-----------------|--------------------|-----|--------|
| | P | A | $maxP$ |
| Train_Small_1 | 12 | 13 | 2 |
| Train_Small_11 | 7 | 8 | 2 |
| Train_Medium_2 | 46 | 47 | 3 |
| Train_Medium_18 | 36 | 37 | 1 |
| Train_Large_1 | 135 | 136 | 4 |
| Train_Large_2 | 127 | 128 | 4 |

Table 4.2: Training Instances

Each algorithm is then run on the set of training instances with various parameters. The following parameter settings are used:

$$p = (0.1, 0.3, 0.5, 1)$$

$$\mu = (0.05, 0.1, 0.3)$$

$$\chi = (0.5, 0.75, 1)$$

$$\sigma = (0.5, 0.75, 1)$$

Each algorithm is run 10 times per parameter configuration (e.g. $p = 0.5, \mu = 0.1, \chi = 1, \sigma = 0.5$), 5 times using crossover-1 and 5 times using crossover-2. Each of these runs goes for 100 generations without any other stopping criteria.

We selected the parameter configuration with the largest number of feasible solutions found after 100 generations. A table showing the percentage of times each parameter configuration finds a feasible solution can be found in Appendix A. From the results

of these runs we find that, for algorithm HA1, parameter configuration with $p = 0.3$, $\mu = 0.05$, $\chi = 0.5$, and $\sigma = 1$, finds the most feasible solution, finding a feasible solution on 91.41% of the runs. For HA2 the parameter configuration with $p = 1$, $\mu = 0.05$, $\chi = 1$, and $\sigma = 0.5$ finds feasible solutions 64.29%, which is the most number feasible solutions for HA2. Parameter configuration with $p = 1$, $\mu = 0.1$, $\chi = 1$, and $\sigma = 0.5$ found the most feasible solutions for GA, finding feasible solutions on 58.49% of the time.

The selected parameters are summarized in the following table, Table 4.3:

| Algorithm | p | μ | χ | σ | % feasible |
|-----------|-----|-------|--------|----------|------------|
| HA1 | 0.3 | 0.05 | 0.5 | 1 | 97.41% |
| HA2 | 1 | 0.05 | 1 | 0.5 | 64.29% |
| GA | 1 | 0.1 | 1 | 0.5 | 58.49% |

Table 4.3: Algorithm Parameters

4.2 Comparison of Algorithms

Using the selected parameters, the algorithms are then evaluated on the remaining instances. There are 15 evaluation instances in total; 5 small instances, 5 medium-sized instances, and 5 large instances. The instances used are shown in Tables 4.4, 4.5, and 4.6.

| Small Instances | | | |
|-----------------|-----|-----|--------|
| Instance | P | A | $maxP$ |
| Small_3 | 14 | 15 | 3 |
| Small_4 | 10 | 11 | 4 |
| Small_6 | 5 | 6 | 3 |
| Small_13 | 11 | 12 | 1 |
| Small_20 | 7 | 8 | 2 |

Table 4.4: Small Instances

| Medium-Sized Instances | | | |
|------------------------|-----|-----|--------|
| Instance | P | A | $maxP$ |
| Medium_2 | 46 | 47 | 3 |
| Medium_4 | 38 | 39 | 4 |
| Medium_5 | 40 | 41 | 1 |
| Medium_10 | 44 | 45 | 2 |
| Medium_11 | 33 | 34 | 3 |

Table 4.5: Medium-Sized Instances

| Large Instances | | | |
|-----------------|-----|-----|--------|
| Instance | P | A | $maxP$ |
| Large_3 | 135 | 136 | 3 |
| Large_4 | 121 | 122 | 1 |
| Large_5 | 131 | 132 | 1 |
| Large_6 | 124 | 125 | 1 |
| Large_7 | 106 | 107 | 4 |

Table 4.6: Large Instances

The small instances have been solved to optimality using the "Chuffed" **CP** solver in MiniZinc [38, 39]. The MiniZinc solver was also able to provide feasible solutions to some of the medium-sized instances within the 2 hour time limit, but these solutions are not necessarily optimal. On the large instances the MiniZinc solver was unable to find any feasible solutions within 2 hours. Therefore, the MiniZinc **CP** results cannot be used as benchmarks for comparison on these large instances.

The Genetic Algorithm and the Memetic Algorithms were run on all three sets of instances with a general stopping criterion of 100 iterations without improvement and running time limit of 2 hours maximum.

4.2.1 Results

Here we provide and examine the results of the algorithms using the selected parameters shown in Table 4.3. Each algorithm is run 20 times per instance: 10 times using crossover-1 and 10 times using crossover-2. The tables in this section use the following abbreviations:

- **GA_C1**: Genetic Algorithm using crossover-1.
- **GA_C2**: Genetic Algorithm using crossover-2.
- **HA1_C1**: Memetic Algorithm HA1 using crossover-1.
- **HA1_C2**: Memetic Algorithm HA1 using crossover-2.
- **HA2_C1**: Memetic Algorithm HA2 using crossover-1.
- **HA2_C2**: Memetic Algorithm HA2 using crossover-2.
- **CP**: MiniZinc "Chuffed" Constraint Programming solver [38, 39] using a **CP** formulation previously defined and implemented by [5].

In the following tables, the * symbol indicates that a feasible solution was not found by that algorithm with that crossover variant and the fitness value reported includes

the deviation penalty value. The [†] symbol indicates that the MiniZinc results are not guaranteed to be optimal. In Table 4.8 and Table 4.11, the % gap in fitness between the algorithms solutions and the CP solver results can be seen. The % gap is defined by the formula

$$\%gap = 100 * \frac{F - CP}{F}$$

where F is the best fitness found by the algorithm being considered and CP is the fitness found by the CP solver.

On the small set of instances we had expected that the three metaheuristic algorithms would find exact results, but this unfortunately was not the case on nearly all the instances. However, all three algorithms found feasible solutions to these instances. We can see the best found fitness of each algorithm and crossover combination in Table 4.7. For these instance, the CP solver was able to find the true optimal solutions to every instance within short running times. For most of the small instances (excluding Small_13 with HA1_C1), the algorithms were not able to reach the true optimal solutions found by the CP solver but all the algorithms found feasible solutions. Table 4.8 shows the percent difference between the best solution found by each algorithm and the exact solution returned by the CP solver. In general, the algorithms performed best on instance Small_13 in terms of fitness, with relatively small gaps. GA found feasible solutions on all of small instance, but the best fitness values reported by GA on many of the small instances were excessively far from the exact solutions with the largest gap of 58.49% on instance Small_3. HA1 and HA2 tended to find more accurate solutions than GA on all the small instances. This can be seen clearly on most instance but the difference is most stark on instances Small_3 and Small_13. On instance Small_3, GA had a gap 58.49% for both crossovers while HA1 has a gap of only 24.14% for both crossovers and HA2 has a gap of 33.33% for crossover-2 and 37.14% for crossover-2. On instance Small_13, GA found solutions with a gap of 10.42% for crossover-1 and 24.56% for crossover-2 while HA1 and HA2 found results with gaps of at most 4.44%.

In Table 4.9, we can see the average running times of the algorithm and crossover combinations on the small instances. We can see that on these small instances the algorithms were able to find solutions within less than a second. In the final column of this table the solve time of the CP solver can be seen. Here we see that on many of the instances the CP solver found solutions in time similar to the Genetic and Memetic Algorithms. On instance Small_3 and instance Small_4 the CP solver took considerably longer than the metaheuristic algorithms; on instance Small_3 the CP solver took 6 minutes and 25 seconds to find the true optimal solution, where the worst average running time of the metaheuristic algorithms was 0.32 seconds and the best average running time of the metaheuristic algorithms was 0.14s. However, the solutions found by the metaheuristics in these running time are feasible yet not exact. For instance Small_4, the CP solver took 1 minute and 8 seconds to find the exact solution while the other algorithms took fractions of a second to find only a feasible feasible solution that was not exact. Although the running times are shorter, the CP solver guarantees exact solutions, and the extra few minutes of running time can be worth the guarantee of exact solutions.

| Instance | GA_C1 | GA_C2 | HA1_C1 | HA1_C2 | HA2_C1 | HA2_C2 | CP |
|----------|-------|-------|-----------|--------|--------|--------|-----------|
| Small_13 | 48 | 57 | 43 | 44 | 44 | 45 | 43 |
| Small_20 | 35 | 31 | 34 | 32 | 32 | 31 | 27 |
| Small_3 | 53 | 53 | 29 | 29 | 33 | 35 | 22 |
| Small_4 | 24 | 27 | 23 | 20 | 20 | 22 | 17 |
| Small_6 | 11 | 10 | 11 | 9 | 8 | 9 | 6 |

Table 4.7: Best fitness on small instances

| Instance | % Gap | | | | | |
|----------|--------|---------------|---------------|---------------|---------------|---------------|
| | GA_C1 | GA_C2 | HA1_C1 | HA1_C2 | HA2_C1 | HA2_C2 |
| Small_13 | 10.42% | 24.56% | 0.00% | 2.27% | 2.27% | 4.44% |
| Small_20 | 22.86% | 12.90% | 20.59% | 15.62% | 15.62% | 12.90% |
| Small_3 | 58.49% | 58.49% | 24.14% | 24.14% | 33.33% | 37.14% |
| Small_4 | 29.17% | 37.04% | 26.09% | 15.00% | 15.00% | 22.73% |
| Small_6 | 45.45% | 40.00% | 45.45% | 33.33% | 25.00% | 33.33% |

Table 4.8: % Gap between best fitness and CP fitness results on small instances

| Instance | GA_C1 | GA_C2 | HA1_C1 | HA1_C2 | HA2_C1 | HA2_C2 | CP |
|----------|-------|-------|--------------|--------------|--------|--------|-----------|
| Small_13 | 0.18s | 0.14s | 0.12s | 0.11s | 0.22s | 0.13s | 1.07s |
| Small_20 | 0.11s | 0.10s | 0.09s | 0.09s | 0.12s | 0.10s | 0.31s |
| Small_3 | 0.32s | 0.23s | 0.19s | 0.14s | 0.30s | 0.20s | 6m 24.66s |
| Small_4 | 0.20s | 0.13s | 0.12s | 0.11s | 0.19s | 0.17s | 1m 8.30s |
| Small_6 | 0.09s | 0.09s | 0.09s | 0.08s | 0.10s | 0.09s | 0.15s |

Table 4.9: Average running times on small instances

On the set of medium-sized instances, the Memetic Algorithms are able to find feasible solutions on all the medium-sized instances. Furthermore, HA1 and HA2 begin to outperform the CP solver in some areas. The CP solver was only able to solve 2 of the instances within the time limit of 2 hours, and these solutions are not guaranteed to be exact. Table 4.10 shows the best fitness found by all the algorithms and crossover variant combinations on the medium-sized instance. Here the best fitness values found for each instance are highlighted in bold. Table 4.11 shows the percent gap in best fitness between the solutions found by the CP solver and the solutions found by each algorithm and crossover variant. This value is left blank if the CP solver could not provide any solutions on that instance. Table 4.12 shows both the solve time of the CP solver and the average running time of each algorithm and crossover variant on the medium-sized instances. In this table the shortest running times are highlighted in bold. However, these short running times do not necessarily provide feasible solutions, therefore the

shortest running times that provide feasible solutions have also been highlighted in red.

For medium-sized instances Medium_2, Medium_20 and Medium_4, the CP solver was unable to find a solution within the running time limit of 2 hours. On the remaining medium-sized instance, the CP solver found feasible solutions within the running time limit but these solutions are not guaranteed to be the true optimum since the solver returns the best solution found so far when the time limit of 2 hours is reached. In Tables 4.10 and 4.11 one can see that on the two instance where the CP solver provided a solution, both Ha1 and HA2 provide feasible solutions that have smaller fitness than the CP solutions, and we can see in Table 4.12 that these solutions are provided in much shorter running times than the CP solutions. GA was able to find feasible solutions on several of the medium-sized instances, but not consistantly, with crossover-1 on instnace Medium_20 and with crossover-2 on instance Medium_10, GA could not provide feasible solutions. Furhtermore, unlike HA1 and HA2 the solutions provided by GA on instances Medium_10 and Medium_11 that are feasible are not better than the CP solver solutions. We can see in table 4.10 that both Memetic Algorithms start to outperform the Genetic Algorithm. HA1 and HA2 could consistently find solutions to all instances with either crossover, but GA could not consistently do the same. Furthermore, where GA did find feasible solutions, the solutions found by HA1 and HA2 on the same instance had far better fitness values than the solutions found by GA. We can also see in Tables 4.10 and 4.11 that HA1 even outperforms HA2 on the medium-sized instances since HA1 tends to find solutions with better fitness. We discuss this further when we examine the values in Table 4.10 and Table 4.11 in more detail.

We can see in Table 4.12 that the running times of HA1 and HA2 are somewhat similar. We expect HA1 to have longer running times than HA2 since the Hill Climbing procedure is applied as every generation. However, the population parameter for HA1 is $p = 0.3$, which leads to a much smaller population size than $p = 1$ used with HA2. A much smaller population leads to faster running times. We can see in Table 4.12 that HA1 and HA2 outperform the CP solver in terms of running time as well. We can see in this table and Tables 4.10 and 4.11 that HA1 and HA2 not only find better solutions than the CP solver, but the average running time of these algorithms is also far superior to that of the CP solver. On the medium sized instances, GA also had running times similar to HA1 nad HA2, but the solutions provided by GA were not better than the CP solver and in some cases GA could not provide feasible solutions. Thus, for the medium-sized instance, CP may be a better algorithm than GA. In the table of running times, Table 4.12, we can also see that crossover-2 gives much faster running times than crossover-1 on all algorithms. For every instance, every algorithm using crossover-2 has a faster average running time than that same algorithm using crossover-1.

Several conclusions can be drawn when considering the fitness in Table 4.10 and the fitness gaps in Table 4.11. In Table 4.11 we can see just how much better the solutions found by HA1 are to the CP solutions. HA1 with crossover-1 provides a solutions 68.25% better than the CP result on instance Medium_10 and 71.43% better on instance Medium_11. HA1 with crossover-2 gave a best fitness that is 69.60% better than the CP solver on

instance Medium_10 and 65.52% better than the CP solver on instance Medium_11. Furthermore, the algorithm that provided these solutions had average running times of only a few minutes. HA1 clearly outperformed both GA and HA2 on the medium-sized instances with HA1 finding solutions with a far better fitness value than the feasible solutions found by HA2 and GA. Additionally, GA could not consistently find feasible solutions with both crossovers. On every medium-sized instance, HA1 found the best fitness values out of all the algorithms. We can see the best fitness over all highlighted in bold in Table 4.10 and all the emboldened results belong to HA1. These results are promising; the Memetic Algorithms (HA1 and HA2) are able to provide feasible solutions that are often far superior to the solutions found by the CP solver within a matter of seconds, while the CP solver was only able to solve two of the instances within its time limit of 2 hours.

| Instance | GA_C1 | GA_C2 | HA1_C1 | HA1_C2 | HA2_C1 | HA2_C2 | CP |
|-----------|-------|-------|-----------|------------|--------|--------|------------------|
| Medium_10 | 275 | 2897* | 126 | 125 | 193 | 195 | 212 [†] |
| Medium_11 | 174 | 175 | 84 | 87 | 122 | 136 | 144 [†] |
| Medium_2 | 175 | 217 | 70 | 75 | 130 | 135 | — |
| Medium_20 | 1292* | 291 | 106 | 101 | 200 | 195 | — |
| Medium_4 | 274 | 291 | 119 | 114 | 157 | 187 | — |

Table 4.10: Best fitness on medium-sized instances

| Instance | % Gap | | | | | |
|-----------|--------|---------|----------------|----------------|---------|--------|
| | GA_C1 | GA_C2 | HA1_C1 | HA1_C2 | HA2_C1 | HA2_C2 |
| Medium_10 | 22.91% | 92.68%* | -68.25% | -69.60% | -9.84% | -8.72% |
| Medium_11 | 17.24% | 17.71% | -71.43% | -65.52% | -18.03% | -5.88% |
| Medium_2 | — | — | — | — | — | — |
| Medium_20 | — | — | — | — | — | — |
| Medium_4 | — | — | — | — | — | — |

Table 4.11: % Gap between best fitness and CP fitness results on medium-sized instances

| Instance | GA_C1 | GA_C2 | HA1_C1 | HA1_C2 | HA2_C1 | HA2_C2 | CP |
|-----------|--------|---------------|--------|--------------|--------|--------------|----|
| Medium_10 | 3.89s | 1.84s* | 3.24s | 1.99s | 4.42s | 1.90s | 2h |
| Medium_11 | 1.55s | 0.69s | 1.01s | 0.63s | 1.58s | 0.74s | 2h |
| Medium_2 | 4.12s | 1.89s | 3.09s | 2.08s | 4.42s | 1.81s | 2h |
| Medium_20 | 3.55s* | 1.98s | 3.88s | 2.43s | 5.01s | 1.89s | 2h |
| Medium_4 | 2.23s | 1.27s | 1.85s | 1.03s | 2.25s | 1.23s | 2h |

Table 4.12: Average running times on medium-sizes instances

In Tables [4.13](#) and [4.14](#) we can see the results of the algorithms on the set of large instances. The CP solver was unable to provide any results to any of these instances within the time limit of 2 hours. Therefore, we cannot compare the results of the Genetic and Memetic Algorithms to the CP results. However, we can compare the algorithms to each other and assess the best fitness values and running times of the results of the algorithms on the large instances. In Table [4.13](#), it can be seen that HA1 clearly outperforms HA2 and GA in terms of fitness by a large margin. HA1 was able to find feasible solutions to all the large instances, while GA found no feasible solutions and HA2 was only able to find feasible solutions on a few of the instances. GA and the CP solver were both unable to find feasible solutions on these large instances. HA2 found feasible results on several instance (Large_4, Large_6, and Large_7), but the fitness values of these solutions were far greater than that of the HA1 results on the same instances. This can be seen clearly on instance Large_7 where the best fitness found by HA2 is 509 when crossover-1 is used and 542 when crossover-2 is used. However on the same instance, HA1 found solutions with fitness values as low as 214 using crossover-1 and 194 using crossover-2 on the same instance.

In Table [4.14](#), the fastest average running times are highlighted in bold, but, as in Table [4.12](#), these do not necessary return feasible results. Thus, the algorithms with the best average running times that also return feasible results are highlighted in red. In this table (Table [4.14](#)) we see that the average running times of both the Genetic and Memetic Algorithms tend to be only a few minutes despite the instances being very large. On these large instances, we also begin to see that HA1 does have longer running times than HA2 and GA. But this is a difference of only a few minutes at most. For GA and to some extent HA2 the reported running times may seem good but these algorithms did not provide feasible solutions so one cannot claim that these average running times are superior to algorithms with longer average running times that also provide feasible solutions. HA1 was able to find feasible solutions to all the large instances and although the average running times of HA1 are slightly longer than the running times of HA2 and GA, GA and HA2 did not consistently find feasible solutions. Here HA1 outperforms both GA and HA2 by a large margin since feasible solutions are consistently found within a few minutes. The longest average running time was HA1 using crossover-1 on instance Large_3, which took 7 minutes and 21 seconds. This is superior to the CP solver since the CP solver found no results in 2 hours while HA1 found feasible results within a few minutes. Taking a look at the crossover variants' average running times in Table [4.14](#), we see again that crossover-2 is faster than crossover-1 on all instances and with all algorithms.

As expected, the performance of the two crossover variants is dependent on the instance. From Tables [4.9](#), [4.12](#), and [4.14](#), it can be seen that over all sets of instances (excluding a few instances in the small set where the crossovers had the same average running time) crossover-1 increases the running times of all the algorithms. This is to be expected since crossover-1 tends to have more exchange operations than crossover-2. For the large instances in Table [4.13](#), it is difficult to determine which crossover may be better in terms

| Instance | GA_C1 | GA_C2 | HA1_C1 | HA1_C2 | HA2_C1 | HA2_C2 | CPs |
|----------|---------|---------|------------|------------|--------|--------|-----|
| Large_3 | 153292* | 200417* | 310 | 286 | 31234* | 34013* | — |
| Large_4 | 5914* | 13500* | 429 | 428 | 698 | 692 | — |
| Large_5 | 29587* | 55676* | 482 | 483 | 8668* | 11229* | — |
| Large_6 | 27857* | 30346* | 467 | 468 | 5621* | 726 | — |
| Large_7 | 37943* | 31356* | 214 | 194 | 509 | 542 | — |

Table 4.13: Best fitness on large instances

| Instance | GA_C1 | GA_C2 | HA1_C1 | HA1_C2 | HA2_C1 | HA2_C2 | CP |
|----------|---------|---------|--------|---------------|---------|---------------|----|
| Large_3 | 2m 15s* | 1m 8s* | 7m 21s | 3m 43s | 2m 36s* | 1m 7s* | 2h |
| Large_4 | 1m 1s* | 44s* | 1m 48s | 52s | 1m 7s | 28s | 2h |
| Large_5 | 2m 7s* | 1m 17s* | 3m 44s | 1m 55s | 2m 28s* | 1m* | 2h |
| Large_6 | 1m 17s* | 34s* | 2m 3s | 57s | 52s* | 48s | 2h |
| Large_7 | 44s* | 18s* | 1m 18s | 48s | 42s | 22s | 2h |

Table 4.14: Average running times on large instances

of best fitness. On three of the five large instances, crossover-2 provides the best fitness found, but crossover-1 finds the best solutions on the other two instances. The same results can be seen on the medium-sized instances in Table 4.10. Crossover-2 provides the best solution on three of the five instance, while crossover-1 provides the best solution on the rest. Which crossover produces better solutions largely depends on both the instance and the algorithm. However, it is clear from Tables 4.9, 4.12, 4.14 that crossover-2 is far faster than crossover-1 and nothing suggests that it provides less accurate solutions than crossover-1. Therefore, it may be the better crossover overall.

Overall, HA1 produces the best results. Despite having slightly longer running times, HA1 was able to find feasible results to several large instances where HA2 and GA could not. HA1 found feasible solutions to every instances in all three sets, this was not the case for GA and HA2. Furthermore, HA1 found feasible solutions that were superior to the solutions found by the CP solver on the medium-sized instances and found solutions that were superior to even the feasible solutions found by HA2 on the large instances. HA1 was also able to find feasible solutions to all the large instances within a matter of minutes while the CP solver could not find any solution within 2 hours.

The findings are summarized as follows:

- On several small instance the CP solver provided better solutions than the Genetic and Memetic Algorithms within similar solve times.
- HA1 and HA2 performed similarly on the small instances. Both had similar average running times and both provided solutions better than the solutions found by the

GA.

- HA1 produces the best fit solutions on the large and medium-sized instances.
- HA1 found feasible solutions on more instances than HA2 and GA.
- HA1 had slightly longer running times than HA2 and GA on the large instances.
- GA did not find feasible solutions to any of the large instances.
- HA1 found feasible solutions to all of the large instances with average running times of less than 8 minutes.
- For every metaheuristic algorithm, crossover-1 had longer running times than crossover-2.
- The performance of crossover-1 verse crossover-2 is largely dependent on the instance; neither preformed significantly better than the other in terms of fitness value.

Conclusion

5.1 Summary of Work

In this thesis, we introduced a new real-world crowdsourced VRP and we addressed the question of whether we can develop an algorithm that can find good solutions to this problem within reasonable running times. We developed a Genetic Algorithm and two Memetic Algorithms and we produced highly promising results for real-world situations and the delivery industry. An extensive review of the current related work on similar VRP problems was discussed. Metaheuristics on OVRP with capacity and deviation constraints was reviewed as well as a more recent problem using ad-hoc drivers called the VRPOD. Lastly, we review recent Memetic Algorithms developed for various OVRP problems. Two Memetic Algorithms were developed both combining a Genetic Algorithm with Randomized Hill Climbing. In the first algorithm, HA1, the Hill Climbing step was done at every generation. In the second algorithm, HA2, the Hill Climbing step was done only once at the end of the algorithm. We also developed a new mutation operation and a crossover operation where two variations were used. We use these operations in both the Genetic Algorithm and the Memetic Algorithms. We used a Roulette Wheel selection strategy and a replacement strategy where the new children replace the chromosome most similar to them in the previous population. A new solution representation was also developed that was used in both the Genetic Algorithm components and the Randomized Hill Climbing components of our algorithms.

Upon comparing the algorithms against Constraint Programming results and against each other, we found that a Memetic approach finds feasible but not guaranteed optimal solutions in a short amount time. The algorithms were compared to Constraint Programming results on small instances. It was found that Genetic and Memetic Algorithms are faster than the Constraint Programming approach, but do not provide exact solutions as the Constraint Programming solver does. Comparing the algorithms to each other on larger instances, we found that the Memetic Algorithms found better solutions than the

Genetic Algorithm on its own and that HA1 performed the best out of the two Memetic Algorithms. This shows that the Local Search has a positive impact on the the results of the algorithm, especially when it is applied at every generation. The two crossover variants were also compared. Crossover-2 was notably faster on most instances, and the fitness values found using the two crossovers were similar.

The Memetic approach in HA1 was very promising; although the solutions of HA1 are not guaranteed to be exact, the algorithm is able to provide feasible solutions in short running times and the solutions were notably better than other solutions found by the GA and HA2. Although the solutions are not guaranteed to be exact, the fact that they can be provided quickly can be very useful in real-world situations.

5.2 Future Work

In the future, our problem could be expanded to mimic real-world situations even further. For example, time window constraints and multi-depot options could be considered. Another interesting idea to explore to to including the drivers' starting points into the route calculations. In the current setting, the drivers are assumed to start at the depot, but it could be to include their starting locations before they arrive at the depot. There is also much work to be done on the analysis of the cost-effectiveness and emission reductions of this problem. Other variants of the problem where drivers are compensated in a different payment scheme could be considered. A different payment strategy could lead to a multi-objective problem where both route distances and monetary costs are considered. Concerning the algorithms, there is much that could be considered in future work. For example, other hybridization techniques could be considered. Furthermore, the possibility of using a smart population initialization strategy instead of a random initialization could lead to interesting results. For example, selecting packages that are within some distance of the driver's destination first before considering packages further away. There could also be more research done on the two crossover operations, for example experimenting with the crossovers on widely varying instances could help determine what types of instances the crossovers are better suited for. Another interesting area of research would be to explore the effects of the number of iterations of the Local Search in the Memetic Algorithms on the fitness and running times of the algorithms. Since HA1 was able to find feasible solutions within only a few minutes on even the largest instances, it would be interesting to see how much better the fitness values could be if the number of iterations of the Hill Climbing component was increased.

APPENDIX A

This section contains an additional table showing the percentage of times each parameter configuration finds a feasible solution for all three algorithms. The highest percentage of feasible solutions is highlighted in bold for each algorithm.

| percent feasible per parameter configuration | | | | | | |
|--|-------|--------|----------|--------|--------|--------|
| p | μ | χ | σ | GA | HA1 | HA2 |
| 0.1 | 0.05 | 0.5 | 0.5 | 5.45% | 73.58% | 59.26% |
| 0.1 | 0.05 | 0.5 | 0.75 | 14.29% | 82.69% | 55.77% |
| 0.1 | 0.05 | 0.5 | 1 | 11.54% | 80.00% | 53.85% |
| 0.1 | 0.05 | 0.75 | 0.5 | 11.32% | 81.48% | 55.36% |
| 0.1 | 0.05 | 0.75 | 0.75 | 14.55% | 77.36% | 58.49% |
| 0.1 | 0.05 | 0.75 | 1 | 12.96% | 74.51% | 52.94% |
| 0.1 | 0.05 | 1 | 0.5 | 15.38% | 79.63% | 53.85% |
| 0.1 | 0.05 | 1 | 0.75 | 7.41% | 81.13% | 53.85% |
| 0.1 | 0.05 | 1 | 1 | 12.73% | 77.36% | 56.36% |
| 0.1 | 0.1 | 0.5 | 0.5 | 9.43% | 75.47% | 60.38% |
| 0.1 | 0.1 | 0.5 | 0.75 | 23.53% | 85.45% | 60.00% |
| 0.1 | 0.1 | 0.5 | 1 | 22.64% | 79.63% | 60.38% |
| 0.1 | 0.1 | 0.75 | 0.5 | 14.81% | 77.36% | 54.39% |
| 0.1 | 0.1 | 0.75 | 0.75 | 16.07% | 74.51% | 55.56% |
| 0.1 | 0.1 | 0.75 | 1 | 20.37% | 76.36% | 58.49% |
| 0.1 | 0.1 | 1 | 0.5 | 16.98% | 78.85% | 57.69% |
| 0.1 | 0.1 | 1 | 0.75 | 22.22% | 79.25% | 61.11% |
| 0.1 | 0.1 | 1 | 1 | 18.52% | 84.31% | 55.56% |
| 0.1 | 0.3 | 0.5 | 0.5 | 21.15% | 74.07% | 52.94% |
| 0.1 | 0.3 | 0.5 | 0.75 | 30.19% | 86.79% | 58.82% |
| 0.1 | 0.3 | 0.5 | 1 | 18.87% | 78.43% | 54.90% |
| 0.1 | 0.3 | 0.75 | 0.5 | 16.98% | 73.08% | 55.77% |
| 0.1 | 0.3 | 0.75 | 0.75 | 20.37% | 81.82% | 56.60% |

| | | | | | | |
|-----|------|------|------|--------|---------------|--------|
| 0.1 | 0.3 | 0.75 | 1 | 20.37% | 75.47% | 61.11% |
| 0.1 | 0.3 | 1 | 0.5 | 14.81% | 81.48% | 60.38% |
| 0.1 | 0.3 | 1 | 0.75 | 20.00% | 79.25% | 55.77% |
| 0.1 | 0.3 | 1 | 1 | 25.00% | 82.35% | 57.41% |
| 0.3 | 0.05 | 0.5 | 0.5 | 22.22% | 75.00% | 58.18% |
| 0.3 | 0.05 | 0.5 | 0.75 | 22.64% | 83.64% | 50.94% |
| 0.3 | 0.05 | 0.5 | 1 | 20.00% | 90.74% | 62.96% |
| 0.3 | 0.05 | 0.75 | 0.5 | 22.64% | 79.63% | 58.49% |
| 0.3 | 0.05 | 0.75 | 0.75 | 16.07% | 74.07% | 56.00% |
| 0.3 | 0.05 | 0.75 | 1 | 23.08% | 88.89% | 58.18% |
| 0.3 | 0.05 | 1 | 0.5 | 27.45% | 81.48% | 60.00% |
| 0.3 | 0.05 | 1 | 0.75 | 28.30% | 74.07% | 60.38% |
| 0.3 | 0.05 | 1 | 1 | 30.19% | 79.63% | 52.83% |
| 0.3 | 0.1 | 0.5 | 0.5 | 23.08% | 78.43% | 57.89% |
| 0.3 | 0.1 | 0.5 | 0.75 | 21.82% | 78.85% | 57.41% |
| 0.3 | 0.1 | 0.5 | 1 | 30.19% | 86.27% | 60.38% |
| 0.3 | 0.1 | 0.75 | 0.5 | 13.21% | 79.25% | 60.38% |
| 0.3 | 0.1 | 0.75 | 0.75 | 22.22% | 80.77% | 58.18% |
| 0.3 | 0.1 | 0.75 | 1 | 26.92% | 75.47% | 58.49% |
| 0.3 | 0.1 | 1 | 0.5 | 28.85% | 86.27% | 56.00% |
| 0.3 | 0.1 | 1 | 0.75 | 26.92% | 83.02% | 58.82% |
| 0.3 | 0.1 | 1 | 1 | 21.57% | 81.13% | 58.93% |
| 0.3 | 0.3 | 0.5 | 0.5 | 32.08% | 77.36% | 57.41% |
| 0.3 | 0.3 | 0.5 | 0.75 | 26.42% | 79.25% | 56.60% |
| 0.3 | 0.3 | 0.5 | 1 | 25.45% | 81.13% | 55.77% |
| 0.3 | 0.3 | 0.75 | 0.5 | 26.92% | 83.02% | 57.41% |
| 0.3 | 0.3 | 0.75 | 0.75 | 24.07% | 77.78% | 58.49% |
| 0.3 | 0.3 | 0.75 | 1 | 30.77% | 82.69% | 55.56% |
| 0.3 | 0.3 | 1 | 0.5 | 29.63% | 76.79% | 58.49% |
| 0.3 | 0.3 | 1 | 0.75 | 32.08% | 82.14% | 56.60% |
| 0.3 | 0.3 | 1 | 1 | 30.91% | 80.77% | 56.60% |
| 0.5 | 0.05 | 0.5 | 0.5 | 33.93% | 79.25% | 61.54% |
| 0.5 | 0.05 | 0.5 | 0.75 | 30.19% | 76.36% | 61.11% |
| 0.5 | 0.05 | 0.5 | 1 | 32.69% | 78.43% | 62.96% |
| 0.5 | 0.05 | 0.75 | 0.5 | 32.69% | 78.85% | 59.26% |
| 0.5 | 0.05 | 0.75 | 0.75 | 35.09% | 84.91% | 59.26% |
| 0.5 | 0.05 | 0.75 | 1 | 32.08% | 78.85% | 57.41% |
| 0.5 | 0.05 | 1 | 0.5 | 26.42% | 76.47% | 55.56% |
| 0.5 | 0.05 | 1 | 0.75 | 43.40% | 83.64% | 61.54% |
| 0.5 | 0.05 | 1 | 1 | 34.55% | 77.36% | 58.49% |
| 0.5 | 0.1 | 0.5 | 0.5 | 33.96% | 83.33% | 59.62% |
| 0.5 | 0.1 | 0.5 | 0.75 | 41.51% | 84.91% | 57.41% |
| 0.5 | 0.1 | 0.5 | 1 | 39.62% | 76.47% | 60.38% |

| | | | | | | |
|-----|------|------|------|---------------|--------|---------------|
| 0.5 | 0.1 | 0.75 | 0.5 | 31.48% | 86.79% | 59.62% |
| 0.5 | 0.1 | 0.75 | 0.75 | 38.00% | 78.85% | 62.26% |
| 0.5 | 0.1 | 0.75 | 1 | 42.59% | 78.85% | 58.18% |
| 0.5 | 0.1 | 1 | 0.5 | 27.27% | 79.25% | 55.77% |
| 0.5 | 0.1 | 1 | 0.75 | 45.28% | 75.00% | 59.26% |
| 0.5 | 0.1 | 1 | 1 | 41.51% | 75.00% | 58.49% |
| 0.5 | 0.3 | 0.5 | 0.5 | 38.18% | 75.47% | 60.38% |
| 0.5 | 0.3 | 0.5 | 0.75 | 35.85% | 78.57% | 60.38% |
| 0.5 | 0.3 | 0.5 | 1 | 45.45% | 87.27% | 51.85% |
| 0.5 | 0.3 | 0.75 | 0.5 | 38.89% | 81.82% | 61.11% |
| 0.5 | 0.3 | 0.75 | 0.75 | 45.10% | 81.82% | 62.26% |
| 0.5 | 0.3 | 0.75 | 1 | 35.85% | 82.14% | 61.82% |
| 0.5 | 0.3 | 1 | 0.5 | 37.74% | 82.35% | 61.11% |
| 0.5 | 0.3 | 1 | 0.75 | 33.33% | 78.57% | 61.82% |
| 0.5 | 0.3 | 1 | 1 | 34.62% | 80.00% | 61.11% |
| 1 | 0.05 | 0.5 | 0.5 | 48.08% | 79.63% | 59.62% |
| 1 | 0.05 | 0.5 | 0.75 | 50.94% | 76.92% | 62.26% |
| 1 | 0.05 | 0.5 | 1 | 46.30% | 73.08% | 61.54% |
| 1 | 0.05 | 0.75 | 0.5 | 47.27% | 76.92% | 61.54% |
| 1 | 0.05 | 0.75 | 0.75 | 48.15% | 86.79% | 61.54% |
| 1 | 0.05 | 0.75 | 1 | 47.37% | 76.47% | 58.82% |
| 1 | 0.05 | 1 | 0.5 | 50.00% | 81.13% | 64.29% |
| 1 | 0.05 | 1 | 0.75 | 49.06% | 82.14% | 58.82% |
| 1 | 0.05 | 1 | 1 | 52.83% | 79.25% | 62.26% |
| 1 | 0.1 | 0.5 | 0.5 | 50.00% | 87.27% | 58.49% |
| 1 | 0.1 | 0.5 | 0.75 | 50.94% | 77.78% | 59.62% |
| 1 | 0.1 | 0.5 | 1 | 47.06% | 76.92% | 60.00% |
| 1 | 0.1 | 0.75 | 0.5 | 49.06% | 74.07% | 60.00% |
| 1 | 0.1 | 0.75 | 0.75 | 48.15% | 84.62% | 56.86% |
| 1 | 0.1 | 0.75 | 1 | 50.94% | 83.93% | 60.00% |
| 1 | 0.1 | 1 | 0.5 | 58.49% | 80.39% | 63.64% |
| 1 | 0.1 | 1 | 0.75 | 47.27% | 83.64% | 58.18% |
| 1 | 0.1 | 1 | 1 | 51.85% | 76.92% | 58.49% |
| 1 | 0.3 | 0.5 | 0.5 | 52.83% | 87.04% | 61.82% |
| 1 | 0.3 | 0.5 | 0.75 | 53.85% | 77.36% | 61.11% |
| 1 | 0.3 | 0.5 | 1 | 57.69% | 88.68% | 59.62% |
| 1 | 0.3 | 0.75 | 0.5 | 50.00% | 87.27% | 61.54% |
| 1 | 0.3 | 0.75 | 0.75 | 50.00% | 80.00% | 60.38% |
| 1 | 0.3 | 0.75 | 1 | 53.85% | 88.68% | 60.38% |
| 1 | 0.3 | 1 | 0.5 | 54.00% | 81.82% | 62.96% |
| 1 | 0.3 | 1 | 0.75 | 56.36% | 84.62% | 58.93% |
| 1 | 0.3 | 1 | 1 | 51.85% | 84.00% | 60.71% |

List of Figures

| | |
|---------------------------------------|----|
| 3.1 Crossover operation: first steps | 20 |
| 3.2 Crossover operation: final result | 20 |
| 3.3 HA1 | 27 |
| 3.4 HA2 | 27 |

List of Tables

| | |
|--|----|
| 4.1 Instance Generation Value Ranges | 30 |
| 4.2 Training Instances | 31 |
| 4.3 Algorithm Parameters | 32 |
| 4.4 Small Instances | 32 |
| 4.5 Medium-Sized Instances | 32 |
| 4.6 Large Instances | 33 |
| 4.7 Best fitness on small instances | 35 |
| 4.8 % Gap between best fitness and CP fitness results on small instances | 35 |
| 4.9 Average running times on small instances | 35 |
| 4.10 Best fitness on medium-sized instances | 37 |
| 4.11 % Gap between best fitness and CP fitness results on medium-sized instances | 37 |
| 4.12 Average running times on medium-sizes instances | 37 |
| 4.13 Best fitness on large instances | 39 |
| 4.14 Average running times on large instances | 39 |

Bibliography

- [1] G. Bensinger, “Amazon’s next delivery drone: You,” *The Wall Street Journal*, june 2015. [Online]. Available: <https://www.wsj.com/articles/amazon-seeks-help-with-deliveries-1434466857>
- [2] “Amazon flex.” [Online]. Available: <https://flex.amazon.com/>
- [3] E. Morphy, “About walmart’s idea to crowdsource its same-day delivery service,” *Forbes*, 03 2013. [Online]. Available: <https://www.forbes.com/sites/erikamorphy/2013/03/28/about-walmarts-idea-to-crowdsource-its-same-day-delivery-service/?sh=3950d7045e5e>
- [4] Wikipedia contributors, “Wikipedia:about — Wikipedia, the free encyclopedia,” 2022. [Online]. Available: ["https://en.wikipedia.org/w/index.php?title=Wikipedia:About&oldid=1123341785"](https://en.wikipedia.org/w/index.php?title=Wikipedia:About&oldid=1123341785)
- [5] P. T. Shqiponja Ahmetaj, Nysret Musliu, “An exact method for a real-life crowd-sourced package delivery problem,” TU Wien, Tech. Rep. DBAI-TR-2022-126, 2022.
- [6] E. Wirsansky, *Hands-On Genetic Algorithms with Python*. Packt Publishing, 2020.
- [7] N. Krasnogor, *Memetic Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 905–935.
- [8] C. Archetti, M. Savelsbergh, and M. G. Speranza, “The vehicle routing problem with occasional drivers,” *European Journal of Operational Research*, vol. 254, no. 2, pp. 472–480, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0377221716301953>
- [9] G. B. Dantzing and J. H. Ramser, “the truck dispatching problem”, *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.
- [10] A. O. Adewumi and O. J. Adeleke, “A survey of recent advances in vehicle routing problems,” *International Journal of System Assurance Engineering and Management*, vol. 9, no. 1, pp. 155–172, 2018. [Online]. Available: <https://link.springer.com/article/10.1007/s13198-016-0493-4>

- [11] G. Laporte, P. Toth, and D. Vigo, “Vehicle routing: historical perspective and recent contributions,” *EURO Journal on Transportation and Logistics*, vol. 2, no. 1, pp. 1–4, 2013. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2192437620600206>
- [12] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuysse, “The vehicle routing problem: State of the art classification and review,” *Computers Industrial Engineering*, vol. 99, pp. 300–313, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360835215004775>
- [13] D. Sariklis, “Open vehicle routing problem: description, formulations and heuristic methods.” Ph.D. dissertation, London School of Economics and Political Science (University of London), 1997.
- [14] E. Ruiz, V. Soto-Mendoza, A. E. Ruiz Barbosa, and R. Reyes, “Solving the open vehicle routing problem with capacity and distance constraints with a biased random key genetic algorithm,” *Computers & Industrial Engineering*, vol. 133, pp. 207–219, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360835219302694>
- [15] P. Repoussis, C. Tarantilis, O. Bräysy, and G. Ioannou, “A hybrid evolution strategy for the open vehicle routing problem,” *Computers Operations Research*, vol. 37, no. 3, pp. 443–455, 2010, hybrid Metaheuristics. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0305054808002207>
- [16] “Ilog cplex optimization studio documentation 21.1.0,” 2020. [Online]. Available: <https://www.ibm.com/docs/en/icos/20.1.0?topic=variables-cplexinfinity>
- [17] G. Macrina, L. Di Puglia Pugliese, F. Guerriero, and G. Laporte, “Crowd-shipping with time windows and transshipment nodes,” *Computers Operations Research*, vol. 113, p. 104806, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0305054819302485>
- [18] L. Feng, L. Zhou, A. Gupta, J. Zhong, Z. Zhu, K.-C. Tan, and K. Qin, “Solving generalized vehicle routing problem with occasional drivers via evolutionary multitasking,” *IEEE Transactions on Cybernetics*, vol. 51, no. 6, pp. 3171–3184, 2021.
- [19] F. Torres, M. Gendreau, and W. Rei, “Crowdshipping: An open vrp variant with stochastic destinations,” *Transportation Research Part C: Emerging Technologies*, vol. 140, p. 103677, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0968090X22001188>
- [20] L. D. P. Pugliese, D. Ferone, P. Festa, F. Guerriero, and G. Macrina, “Solution approaches for the vehicle routing problem with occasional drivers and time windows,” *Optimization Methods and Software*, vol. 0, no. 0, pp. 1–31, 2022. [Online]. Available: <https://doi.org/10.1080/10556788.2021.2022142>

- [21] D. Yang, “Planning and operation of a crowdsourced package delivery system: Models, algorithms and applications.” Ph.D. dissertation, University of California, Irvine, 2021. [Online]. Available: <https://escholarship.org/uc/item/2kv034hw>
- [22] G. Macrina, L. D. P. Pugliese, and F. Guerriero, “Crowd-shipping: a new efficient and eco-friendly delivery strategy,” *Procedia Manufacturing*, vol. 42, pp. 483–487, 2020, international Conference on Industry 4.0 and Smart Manufacturing (ISM 2019). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2351978920306077>
- [23] D. Yang, M. F. Hyland, and R. Jayakrishnan, “Tackling the crowdsourced delivery problem at scale through a set-partitioning formulation and novel decomposition heuristic,” 2022. [Online]. Available: <https://arxiv.org/abs/2203.14719>
- [24] A. Sampaio, M. Savelsbergh, L. P. Veelenturf, and T. Van Woensel, “Delivery systems with crowd-sourced drivers: A pickup and delivery problem with transfers,” *Networks*, vol. 76, no. 2, pp. 232–255, 2020. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/net.21963>
- [25] Y. Abu Al Hla, M. Othman, and Y. Saleh, “Optimising an eco-friendly vehicle routing problem model using regular and occasional drivers integrated with driver behaviour control,” *Journal of Cleaner Production*, vol. 234, pp. 984–1001, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S095965261932116X>
- [26] R. Y. Fung, R. Liu, and Z. Jiang, “A memetic algorithm for the open capacitated arc routing problem,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 50, pp. 53–67, 2013. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1366554512000932>
- [27] L. Sun, Q. ke Pan, X.-L. Jing, and J.-P. Huang, “A light-robust-optimization model and an effective memetic algorithm for an open vehicle routing problem under uncertain travel times,” *Memetic Computing*, vol. 13, no. 2, pp. 149–167, 2021. [Online]. Available: <https://link.springer.com/article/10.1007/s12293-020-00322-5>
- [28] L. Ran, Z. Jiang, and N. Geng, “A hybrid genetic algorithm for the multi-depot open vehicle routing problem,” *OR Spectrum*, vol. 36, no. 2, pp. 401–421, 2014. [Online]. Available: <https://link.springer.com/article/10.1007/s00291-012-0289-0>
- [29] C. Prins, “Two memetic algorithms for heterogeneous fleet vehicle routing problems,” *Engineering Applications of Artificial Intelligence*, vol. 22, pp. 916–928, 09 2009.
- [30] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1998.
- [31] J.-M. Renders and H. Bersini, “Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways,” in *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, 1994, pp. 312–317 vol.1.

- [32] P. Moscato and C. Cotta, *A Modern Introduction to Memetic Algorithms*. Boston, MA: Springer US, 2010, pp. 141–183. [Online]. Available: https://doi.org/10.1007/978-1-4419-1665-5_6
- [33] T. El-mihoub, A. Hopgood, L. Nolle, and B. Alan, “Hybrid genetic algorithms: A review,” *Engineering Letters*, vol. 3(2), 08 2006.
- [34] B. Wang, W. Wan, and J. B. Birch, “An improved hybrid genetic algorithm with a new local search procedure,” *Journal of Applied Mathematics*, vol. 2013, p. 103591, 2013. [Online]. Available: <https://www.hindawi.com/journals/jam/2013/103591/>
- [35] L. Gharsalli and Y. Guerin, “A hybrid genetic algorithm with local search approach for composite structures optimization,” in *8th European Conference for Aeronautics and Space Sciences. Madrid. AIP publisher*, 2019, p. 9.
- [36] T. El-mihoub, A. Hopgood, L. Nolle, and B. Alan, “Hybrid genetic algorithms: A review,” *Engineering Letters*, vol. 3(2), 08 2006.
- [37] S. Katoch, S. S. Chauhan, and V. Kumar, “A review on genetic algorithm: past, present, and future,” *Multimedia Tools and Applications*, vol. 80, no. 5, pp. 8091–8126, 2021. [Online]. Available: <https://link.springer.com/article/10.1007/s11042-020-10139-6>
- [38] “Minizinc,” 2022. [Online]. Available: <https://www.minizinc.org/>
- [39] G. T. Peter J. Stuckey, Kim Marriott, “The minizinc handbook,” 2018, [Online; accessed 7-November-2022]. [Online]. Available: <https://www.minizinc.org/doc-2.3.0/en/solvers.html>