FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

A Hybrid Evolutionary Algorithm for the Vehicle Routing Problem with Stochastic Demands

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering and Internet Computing

eingereicht von

Sara Pourmanouchehri BSc
Matrikelnummer 0728314

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Univ.-Ass. Dipl.-Ing. Benjamin Biesinger
Univ.-Ass. Dipl.-Ing. Dr.techn. Bin Hu

Wien, 1. Juli 2016

Sara Pourmanouchehri　　Günther Raidl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

A Hybrid Evolutionary Algorithm for the Vehicle Routing Problem with Stochastic Demands

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering and Internet Computing

by

Sara Pourmanouchehri BSc
Registration Number 0728314

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Assistance: Univ.-Ass. Dipl.-Ing. Benjamin Biesinger
Univ.-Ass. Dipl.-Ing. Dr.techn. Bin Hu

Vienna, 1st July, 2016

Sara Pourmanouchehri Günther Raidl

Erklärung zur Verfassung der Arbeit

Sara Pourmanouchehri BSc
Muthgasse 66/212 1190 Wien

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, 1. Juli 2016

Sara Pourmanouchehri

Danksagung

Zunächst möchte ich mich an dieser Stelle bei all denjenigen bedanken, die mich während der Anfertigung dieser Arbeit unterstützt und motiviert haben.
Herrn Professor Dr. Raidl bin ich für seine Unterstützung an dieser Stelle zu tiefem Dank verpflichtet. Mein Dank gilt zudem Herrn Dipl.-Ing. Biesinger und Herrn Dr. Hu. Ohne die unendliche Geduld und die Gründlichkeit von Herrn Biesinger wäre die Fertigstellung dieser Arbeit nicht möglich gewesen.
Darüber hinaus bedanke ich mich bei allen, die durch Korrekturen, Verbesserungsvorschläge und hilfreiche Diskussionen zur Qualität dieser Arbeit beigetragen haben. Gleiches gilt für meine Eltern und Martin, die mir in dieser Zeit eine unersetzbare seelische und moralische Stütze waren.

Acknowledgements

First of all I would like to thank all those who have supported and motivated me during the preparation of this work.
I would like to show my gratitude to my Professor Dr. Raidl for his support. Also many thanks to Dipl.-Ing. Biesinger and Dr. Hu. Finishing this work would have not been possible without patience and thoroughness of sir. Biesinger.
Furthermore, I thank each person who helped me with correction and increasing the quality of this work. The same is true for my parents as well as Martin, who were always with me with their indispensable moral support during this time.

Kurzfassung

Diese Arbeit behandelt eine stochastische Erweiterung des klassischen Routenplanungsproblem, die den Kundenbedarf als Zufallsvariablen betrachtet. Der Schwerpunkt wird auf die Entwicklung eines hybriden evolutionären Algorithmus zur Lösung dieses sogennanten Vehicle Routing Problems mit stochastischem Bedarf (VRPSD) gelegt. Bei solchen Routenplanungsproblemen verlassen die Fahrzeuge das Depot mit voller Fracht, um Kunden zu bedienen, deren exakte Nachfrage jedoch zur Startzeit unbekannt ist und erst bei Ankunft genau feststeht. Das VRPSD ist ein NP-schweres kombinatorisches Optimierungsproblem. Dies bedeutet, dass es unter der Annahme von P$\neq$NP keine Lösung in Polynomialzeit für dieses Problem gibt. Aus diesem Grund wurde ein genetischer Algorithmus entwickelt, der auf der Permutation von Knotenpunkten als Lösungskandidaten beruht, so dass eine möglichst gute Lösung in einem akzeptablen Zeitrahmen gefunden werden kann. Die Evaluierung einer solchen Permutation, die jene Positionen bestimmt, an denen das Fahrzeug wieder befüllt werden muss, beruht auf dynamischer Programmierung. In einem nächsten Schritt wurde eine 2-opt lokale Suche implementiert, um die Suche zu intensivieren. Um die Suche zu beschleunigen, wurde ein Multi-level Evaluierungsschma auf den genetischen Algorithmus angewandt. Mit dieser Methode wurde der Zielfunktionswert eines Lösungskandidaten wiederholt und mit zunehmender Genauigkeit approximiert bis er verworfen oder exakt evaluiert werden konnte. Um ein wiederholtes Evaluieren bereits bekannter Lösungskandidaten zu vermeiden, wurde außerdem ein vollständiges Trie-basiertes Lösungsarchiv implementiert, das Duplikatlösungen in garantiert neue und meist ähnliche Lösungen konvertiert, was diesen Lösungsalgorithmus im Prinzip zu einem exakten Verfahren macht. Die grundsätzliche Idee besteht darin, alle Lösungskandidaten in einer Trie-Datenstruktur zu speichern. Sobald eine neue Lösung generiert wird, wird diese mit dem Lösungsarchiv abgeglichen, um zu garantieren, dass es sich um eine neue Lösung handelt. Sollte die Lösung schon vorhanden sein, produziert das Archiv automatisch eine neue Lösung, die die Duplikatlösung ersetzt statt dieselbe Lösung ein zweites Mal zu betrachten. Die Resultate auf einer Menge von Benchmark Instanzen zeigen, dass das Multi-level Evaluierungsschema die für die Lösungevaluierung benötigte Zeit drastisch reduzieren kann. Die Nutzung des Lösungsarchivs führt hingegen nur in manchen Fällen zu besseren Ergebnissen. Verglichen mit bisher in der Literatur angewandten Algorithmen kann der in dieser Arbeit vorgeschlagene Ansatz daher zu deutlich verbesserten Lösungen des VRPSD beitragen.

Abstract

This work considers a stochastic extension to the classical vehicle routing problem by assuming that the customers' demand are random variables. The focus lies on the development of a hybrid evolutionary algorithm for solving the so-called vehicle routing problem with stochastic demands and preventive restocking (VRPSD). In the VRPSD vehicles leave from a depot with full load, and have to serve a set of customers whose exact demand is only known upon arrival. The VRPSD is an NP-hard combinatorial optimization problem which means that under the assumption that P$\neq$NP there exists no polynomial time solution algorithm. Therefore, a genetic algorithm using the permutation of nodes as solution representation is developed so that an acceptably good solution can be found in reasonable time. The evaluation of such a permutation, which determines the restocking points depending on the current load of the vehicle, is performed by a dynamic programming algorithm. In the next step, to intensify the search, a 2-opt local search based is implemented. Additionally, a multi-level evaluation scheme is applied to the genetic algorithm to speed up the solution evaluations. Using this scheme, the objective value of a solution candidate will be repeatedly approximated with increasing accuracy until it can be discarded or it is evaluated exactly. On top of that, a complete trie-based solution archive is implemented in order to prevent revisiting known solutions and converting them into guaranteed new and usually similar solutions. This makes the overall solution method, in principle, to an exact algorithm. The idea behind is to save all candidate solutions in a trie data structure. Each time a new solution generated, first it is looked up in the solution archive to check whether this solution is already generated or not. In case of revisiting an old solution, a guaranteed new solution candidate will be produced directly by the archive which replaces the duplicate. Computational results on a set of benchmark instances show that the multi-level evaluation scheme is able to drastically reduce the time spent in solution evaluations and that applying the solution archive leads to an improvement only in some instances. Compared to an algorithm of the literature the proposed approach can significantly improve its results.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
1.1 Problem definition 2
1.2 Thesis outline 3

2 Previous Work **5**
2.1 Vehicle Routing Problem 5
2.2 Stochastic Vehicle Routing Problem 6

3 Methods **11**
3.1 Genetic Algorithms 12
3.2 Local Search 14

4 Genetic Algorithm **17**
4.1 Structure of the Genetic Algorithm 17
4.2 Local Search 18
4.3 Multi-level evaluation scheme 22
4.4 Solution Archive 26

5 Computational Results **31**
5.1 Construction of Test 31
5.2 Computational Results 32

6 Conclusions and Future Work **41**
6.1 Future Work 42

List of Figures **43**

List of Tables **44**

CHAPTER 1

Introduction

The Vehicle Routing Problem (VRP) has important applications in the fields of transportation, distribution, and coordination of logistical operations for transporting products. It considers the problem of servicing a number of customers with a fleet of vehicles. In practical contexts, one or several elements of the problem, for example, customers, travel time, or demand may not be known with certainty. However, in practice these values are usually not completely random but follow a known probability distribution [YMB00]. These problems are referred to as stochastic vehicle routing problems (SVRP).

This work concentrates on the vehicle routing problem with stochastic demands and preventive restocking (VRPSD). In the VRPSD vehicles leave from a depot with full load, and have to serve a set of customers whose exact demand is only known upon arrival. The aim is to design delivery or collection routes to minimize the expected cost of the routes. The main goal of this thesis is to develop a state-of-the-art genetic algorithm (GA) for solving the VRPSD and to derive new insights and methods for solving SVRPs in general. More precisely, a set of evaluation and optimization techniques will be developed including local search, dynamic programming, a multi-level evaluation scheme, and a solution archive.

Existing algorithms use a permutation encoding for the visit sequence of the nodes and dynamic programming [YMB00] to evaluate candidate solutions. Due to the large time consumption of evaluating a candidate solution a multi-level evaluation scheme (ML-ES) will be applied. In the ML-ES, the objective value of candidate solutions will be repeatedly approximated level by level with increasing accuracy. On each level, this value is calculated for a vehicle with scaled down capacity and altered probability distributions. As we show in this thesis, the value computed at a particular level is a lower bound to the next level. If this value is also lower bound to the real objective value, we might abort the evaluation earlier. This leads to either an exact solution evaluation or to a situation where the candidate solution can be discarded due to its inferior quality. This will make it possible to reduce the overall time needed for solution evaluations within the GA. The ML-ES is described in more detail in section 4.3.

Finally, a complete solution archive based on a memory-efficient trie data structure will be considered. Here, the aim is to deal with common disadvantages of GAs. One of them is the existence of duplicate candidate solutions which could lead to a loss of diversity and to premature convergence. Applying a solution archive can improve a GA by avoiding (unnecessary) re-evaluations of already assessed candidate solutions, increasing and introducing diversity and reducing the risk of premature convergence. It has already been shown in the literature that a solution archive can improve a GA, especially for problems with compact solution representation and time consuming evaluation [RH10]. In section 4.4 we will show how to apply such a solution archive to the VRPSD.

1.1 Problem definition

Here the problem of stochastic vehicle routing is defined on a complete graph $G = (V, A, C)$, where:

- $V = \{0, 1, ..., n\}$, a set of nodes with node 0 denoting the depot and nodes $1, ..., n$ corresponding to the customers.
- $A = \{(i, j) : i, j \in V . i \neq j\}$, the set of the arcs joining the nodes.
- $D = \{d_{i,j} : i, j \in V . i \neq j\}$, denoting the travel cost (or distance) between nodes i and j.

It is assumed that all customers, the depot included, are fully interconnected. All customers have stochastic demands $\xi_i, i = 1, ..., n$ which are independently distributed with a known distribution and are known only upon arriving at the customer node. It is also assumed that ξ does not exceed the vehicle capacity Q and follows a discrete probability distribution with m possible values $\xi^1, \xi^2, ..., \xi^m$ and the probability mass function $p_{ik} = Prob\left(\xi_k = \xi^m\right), \forall i = 1, ..., n$.

The goal is to find a route that starts and ends at the depot while each customer is visited once. The objective is to minimize the expected cost of the tour by applying an optimization method. The costs under consideration are:

- Cost of traveling from one customer to another as planned.
- Restocking cost; the cost of traveling back to the depot for restocking.

The cost/distance matrix D is symmetric and satisfies the triangular inequality. Each customer i has a density function $p_i(k)$ where $0 \leq k \leq Q$ and Q is the maximum vehicle capacity, denoting the probability of demand k at customer i.

A feasible solution for a single vehicle is a permutation $s = (s(0), s(1), ..., s(n))$ where the first element is always the depot $(s(0) = 0)$, which is called an a priori tour [Spe13]. Then, the computation of the expected costs works as follows.

After visiting a customer j of the a priori tour assume the vehicle has a remaining load of q then $f_j(q)$ denotes the total expected cost from j onward. With this definition, the expected cost of the a priori tour is $f_0(Q)$. Let L_j represent the set of all possible loads at customer j then $f_j(q)$ for $q \in L_j$ satisfies:

$$f_j(q) = min\left\{f_j^p(q), f_j^r(q)\right\} \tag{1.1}$$

where :

$$f_j^p(q) = d_{j,j+1} + \sum_{k:k\leq q} f_{j+1}(q-k)p_{j+1,k} + \sum_{k:k>q} \left[2d_{j+1,0} + f_{j+1}(q+Q-k)\right] p_{j+1,k} \tag{1.2}$$

$$f_j^r(q) = d_{j,0} + d_{j+1,0} + \sum_{k=1}^{k} f_{j+1}(Q-k)p_{j+1,k} \tag{1.3}$$

and the boundary condition $f_n(q) = d_{n,0,q} \in L_n$. In equation 1.2, f_j^p is the expected cost when proceeding directly to the next customer, the third sum describes the case of having not enough capacity left and therefore returning to the depot. In equation 1.3, f_j^r is the expected cost when replacing the direct route with a preventive restock [Spe13]. For every customer j there exists a load threshold h_j that once reached makes a preventive restock the better choice over continuing directly to the next customer as shown by Yang et al. [YMB00].

1.2 Thesis outline

The remainder of this work is organized as follows: Chapter 2 presents a literature review which deals with Stochastic Vehicle Routing Problems or more precisely Vehicle Routing Problems with Stochastic Demands. It also provides a comparison between this work and other literature in this field. Chapter 3 gives an overview of the applied methods. The developed metaheuristics for the VRPSD will be discussed in chapter 4. It consists of a comprehensive description of the GA and the multi-level evaluation scheme. Further, the complete solution archive is introduced. Chapter 5 investigates experimental results.

These results are obtained from the computational tests, which have been run under different configurations. It is concerned with the choice of best configuration in order to achieve the best objective value. Taking the material from the fourth chapter as basis, chapter 6 summarizes and concludes this work. Finally, also in this chapter, an outlook on future work is provided.

CHAPTER 2

Previous Work

A literature review is given below to give a basic understanding of essential subtopics related to this work. First, an introduction to Vehicle Routing Problems (VRPs) is given. Its main principles as well as several constraints that may arise when solving those problems are investigated in section 2.1. Further information about Stochastic Vehicle Routing Problems (SVRPs) and selected variants is given in section 2.2.

2.1 Vehicle Routing Problem

Dantzig and Ramser [DR59] introduced the Vehicle Routing Problem (VRP) in 1960 as a generalization of the Traveling Salesman Problem (TSP). The VRP is used in supply chain management in the physical delivery of goods and services. VRP is concerned with servicing a number of customer with a fleet of vehicles. The implicit goal of this kind of problem is minimizing the cost of distributing the goods.

Many methods have been developed to reach good solutions to the problem. Determining an optimal or near optimal solution for most variants of VRPs is hard, so in practice heuristic and metaheuristic methods have been developed that find acceptable solutions.

The VRP is generally defined on a graph $G = (\nu, E, C)$, where $\nu = \{\nu_0, ..., \nu_n\}$ is the set of vertices; $E = \{(\nu_i, \nu_j) \mid (\nu_j, \nu_i) \in \nu^2, i \neq j\}$ is the arc set. A matrix $C = (C_{i,j})$, $\forall(\nu_i, \nu_j) \in E$ is defined on E. The coefficients $c_{i,j}$ represent distances, travel costs or travel times.

Traditionally, vertex ν_0 represents a depot and the rest represent customer locations that need to be served. The number of vehicles can be a given constant or a decision variable. Each vehicle has the same capacity Q [PGGM12]. The homogeneous VRP consists in finding a set of routes for k identical vehicles based at the depot such that each of the vertices is visited exactly once while satisfying some constraints.

Having a single depot, a homogeneous fleet of vehicles or one route per vehicle are among the assumptions that belong to the basic VRP. These assumptions can be modeled by introducing additional constraints to the problem. Usually, in real world VRPs, many side constraints appear namely Capacitated VRP (CVRP), Multiple Depot VRP [HHJL08], Vehicle Routing Problem with Multiple Trips (VRPMT) [PS03], VRP with Pick-Up and Delivering (VRPPD) [Mos98], Split Delivery VRP [DT90], Stochastic VRP (SVRP), Open Vehicle Routing Problem (OVRP) [SP00], the Heterogeneous fleet VRP (HVRP), and Periodic VRP.

The most common constraints are the following [GLS96a]:

- *Capacity constraints*: each customer ν_i has a demand d_i and the total demand of any route may not exceed the vehicle capacity, vehicles make collections or deliveries at all customers. Delivery and collection problems are symmetrical with one another and equivalent from a modeling point of view. Here, problems will be described in form of collections.
- *Duration constraints*: the total length of each route may not exceed a preset constant L.
- *Time window constraints*: each vertex ν_i must be visited within a time interval $[a_i, b_i]$.
- *Stochastic VRPs* appear whenever one or several components of the problem are random. Applying randomness to any of the above named constraints (capacity, duration) makes the VRP to a SVRP.

The VRP has been extensively studied in the optimization literature because of its wide applicability and its importance in determining efficient strategies for reducing operational cost in distribution networks.

2.2 Stochastic Vehicle Routing Problem

Whenever some elements of the problem are random, Stochastic Vehicle Routing Problems (SVRPs) will arise. SVRPs can be viewed in the context of stochastic programming which is modeled in two stages. At a first stage, a planned or "a priori" solution is determined. The realizations of the random variables are then disclosed and, at a second stage, a recourse or corrective action is applied to the first stage solution. The recourse usually generates a cost or a saving that may have to be considered when designing the first stage solution [GLS96a].

There are different variations and specializations of particular types of SVRPs which can be studied and will be shortly described here [GLS96a].

- *The Traveling Salesman Problem with Stochastic Customers (TSPSC)*: or probabilistic travelling salesman problem (PTSP). In this problem each vertex is present with a specified probability. In the first stage a Hamiltonian tour through all vertices is constructed and the set of present vertices is then revealed. In the second stage solution, the tour is followed by simply skipping absent customers.
- *The Traveling Salesman Problem with Stochastic Travel Times (TSPST)*: Here, times are random variables. It is attempted to determine an a priori solution such that the probability of completing the tour within a given deadline is maximized.
- *The m-Traveling Salesman Problem with Stochastic Travel Times (m-TSPST)*: The only difference of this problem with the previous one is the number of vehicles. Simpler, it is the m vehicle version of TSPST, with all routes starting and ending at a common depot. Here, the number of vehicles is a decision variable with an associated fixed cost.
- *The Vehicle Routing Problem with Stochastic Demands (VRPSD)*: This problem is the most studied of all SVRPs and also considered in this thesis. Here, customer demands are random variables usually assumed to be independent and each customer demand follows a probability distribution. Since the customer demand can be uncertain, a vehicle may arrive at a customer without enough capacity to satisfy its demand and may need to apply a recourse to recover the route's feasibility [MRV15].
- *The Vehicle Routing Problem with Stochastic Customers (VRPSC)*: VRPSC is a direct extension of the TSPSC: Customers are present with some probability but have deterministic demands. The vehicle capacity must be respected and return trips to the depot may be necessary whenever it becomes attained. As in the TSPSC, absent customers are skipped in the second stage solution.
- *The Vehicle Routing Problem with Stochastic Customers and Demands (VRPSCD)*: This problem is a combination of the VRPSC and the VRPSD. In a first stage, one determines a set of routes starting and ending at the depot and visiting each customer exactly once. The set of customers with zero demand is then gradually revealed, but the positive demand of every remaining customer becomes known only when the vehicle arrives at the customer's location. In the second stage, the first stage routes are followed as planned, with the following two exceptions: (1) any absent customer is skipped; (2) whenever the vehicle capacity becomes exceeded, it returns to the depot to unload, and resumes collections starting at the last visited customer; if for any customer the vehicle capacity becomes exactly attained, the vehicle then returns to the depot and resumes collections at the next present customer along its route.

The problem which will be addressed in this thesis is the VRPSD. SVRPs are usually generalizations of VRPs and thus NP-hard problems. An important property of SVRPs is that they have an objective function to compute the expected cost which is usually computationally much more expensive than their deterministic counterparts [CLM01].

In the VRPSD, the stochastic parameter is customer's demand. However, the actual demand is not totally unknown: The potential demand is known but only with a certain probability. Additionally, the vehicle's capacity is limited and due to uncertainty of demand, vehicles may be depleted before the entire demand of the customers has been met. In this case, the vehicle has to return to the depot for restocking.

There exist different restocking policies. According to the simplest policy, the vehicle returns to the depot whenever it cannot (completely) satisfy the demand of the current customer. Whenever this happens the vehicle restocks at the depot and continues the tour at the (partially) unsatisfied customer. This strategy is used in the majority of the work of the literature.

A more efficient but also more complex policy is a preventive restocking. Preventive restocking is introduced by Yang et al. [YMB00]. They have utilized a restocking policy which enables the option of preemptive replenishment of capacity before the actual stockout occurs. The goal of this action is to avoid the bad situation when the vehicle has not enough load to serve a customer and thus it has to perform a back and forth trip to the depot for completing the delivery at the customer. As it was shown that this is the optimal restocking strategy [YMB00], the preventive restocking policy is applied in this thesis. The dynamic programming formulation introduced in section 1.1 describes the computation of the expected costs using this policy.

In the next paragraphs solution methods for VRPSDs, which have been proposed in the literature, are listed and briefly presented.

The underlying GA used in this thesis is based on the work of Sperl [Spe13]. The VRPSD has been comprehensively considered in Yang et. al. [YMB00]. Here, it has been assumed that the demands are uncertain, and actual demand is revealed only upon the visit to the customer. Two heuristic algorithms are developed to construct both single and multiple routes that minimize total travel cost. The author also claims that the heuristic procedures produce quality solutions and are efficient. Going even further, Marinakis et al. [MMM14] use a clonal selection algorithm for the VRPSD. They also combine this algorithm with a Variable Neighborhood Search (VNS), and an Iterated Local Search (ILS) algorithm.

Additionally, Yang et al. [YMB00] pointed to the most important characteristic of the SVRP, which is the uncertainty of demand of each customer. Due to this characteristic, the goods on the vehicle may be depleted at some point along a route before the total demand on the route has been met.

In their work, Yang et al. described preventive restocking, which is to force the vehicle return to the depot for reloading and to return to the route at the point where the out-of-stock occurred [YMB00]. However, this solution is manipulated and the probability of the demand of the following customer plays an important role in taking the decision.

Christiansen et al. [CL07] introduced a new exact algorithm for the capacitated vehicle routing problem with stochastic demands (CVRPSD). The CVRPSD can be formulated as a set partitioning problem and it is shown that the associated column generation subproblem can be solved using a dynamic programming scheme. However in this work, a different restocking strategy has been used.

Shanmugam et al. [SGV11] studied a Particle Swarm Optimization (PSO) and Hybrid PSO (HPSO) approach besides a very basic GA used to solve VRPSD. The Dynamic Programming method by Yang et al. [YMB00] is used to find objective values of candidate solutions generated by the GA, PSO, and HPSO.

Shanmugan et al.[GLS96b] used a tabu search for this kind of problem. They used tabu search and compared the results with known optimal solutions whose size vary from 6 to 46 customers. It has been observed that the heuristics produce an optimal solution in 89.45% of cases, with an average deviation of 0.38% from optimality.

Bianchi et al. [BBC$^+$06] analyzed the performance of metaheuristics on the vehicle routing problem with stochastic demands (VRPSD). Fast approximations of the objective function are therefore appealing because they would allow for an extended exploration of the search space. It has been claimed that two hybridized versions of iterated local search and evolutionary algorithm achieve better solutions than state-of-the-art algorithms.

Additionally a comparison of the performance of five well-known different metaheuristic were performed by Bianchi. These metaheuristics are Simulated Annealing (SA), Tabu Search (TS), Iterated Local Search (ILS), Ant Colony Optimization (ACO), and Evolutionary Algorithm (EA). The results show that iterated local search and evolutionary algorithm attain better solutions than the other algorithms.

Complete solution archives have been used for optimization problems as well. Raidl and Hu [RH10] discuss common weaknesses of GAs and propose a complete solution archive based on a special binary trie structure for GAs with binary representations that efficiently stores all evaluated solutions during the heuristic search.

CHAPTER 3

Methods

The VRPSD is, like most VRPs, an NP-hard problem for which a computationally efficient solution algorithm has neither been found nor shown to be non-existent [BBC$^+$06] (under the assumption that P $\neq$ NP).

Not only the VRPSD but many other combinatorial optimization problems are also NP-hard. This means that a polynomial time solution algorithm does not exist and therefore solution algorithms generally do not scale well with the problem size and usually only small instances can be optimally solved. When the problems involve large instances, it may be hard to solve these problems in reasonable CPU times, which occurs often in real-life optimization problems. They are usually of big size and since exact approaches are inadequate, heuristics are commonly used in practice [SCP$^+$13].

A heuristic is a technique designed for approximately solving a problem more quickly when exact methods are too slow, or for finding an approximate solution when exact methods fail [Ben82].

A metaheuristic is designed to find, generate, or select a heuristic that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity [BDGG09]. Metaheuristics sample a set of solutions which is too large to be completely enumerated. Metaheuristics may make few assumptions about the optimization problem being solved, and so they may be usable for a variety of problems [BR03].

There is a wide selection of metaheuristics such as Simulated Annealing, Tabu Search, Ant Colony Optimization, and Genetic Algorithms. These techniques have been extensively used and their robustness and applications are well established in the VRP literature.

As this thesis strongly build upon a genetic algorithm and local search both of these methods are described next.

3.1 Genetic Algorithms

Genetic algorithms (GAs) are one of the most popular evolutionary algorithms and a majority of optimization problems have been approached by GAs [GH88]. They are search heuristics that produce a set of solutions based on selection and evolution. They provide efficient and effective techniques to generate solutions to optimization and search problems, and machine learning applications using techniques inspired by Darwin's theory about evolution, such as inheritance, mutation, selection, and crossover [Mit98].

GAs derive their behavior from the processes of evolution in nature. This is done by the creation of a population of individuals represented by chromosomes within a machine. Each individual represents a point in a search space and a feasible solution. These individuals follow the genetic analogy. This means that their behavior is linked to chromosomes and the variables are analogous to genes. The chromosome is composed of several genes; accordingly the solution is composed of several variables. A fitness value is assigned to each solution using a fitness function (often the objective function), which enables the individuals to compete with each other to produce offsprings better than the parents by combining information from the chromosomes.

The individuals in the population then go through a process of evolution. This means that solutions from one population are taken and used to form a new population. Solutions which are selected to form new solutions (offsprings) are selected according to their fitness, resembling a survival of the fittest principle. Accordingly, the more suitable they are, the more chances they have to reproduce. This is repeated until a certain condition is satisfied (see Algorithm 3.1).

Algorithm 3.1: A standard GA in pseudocode

```
1 Initialize initialize population ($t$)
2 Evaluate determine fitness of population ($t$)
3 while Termination condition is not satisfied do
4     Select select parents from population ($t$)
5     Crossover perform crossover on parents creating population ($t+1$)
6     Mutate perform mutation of population ($t+1$)
7     Evaluate determine fitness of population ($t+1$)
8     $t = t + 1$
9 end
```

New populations are created during the steps inside the while loop which is shown in algorithm 3.1. Crossover occurs in these steps. It selects genes from parent chromosomes and creates a new offspring. There are different ways to make crossover, e.g. One-point crossover, Two-point crossover, Cut and splice, Uniform Crossover and Half Uniform Crossover, and Arithmetic crossover, that can be rather complicated.

Take **11001**011 as parent-A and 11011**111** as parent-B. In One-point crossover, one crossover point is selected and a binary string from the beginning of the chromosome to the crossover point is copied from one parent. The rest is copied from the second parent. So parent-A + parent-B is equal to **11001**011 + 11011**111** = **11001111**.

By just applying crossover the diversity of the population may decrease. Therefore, the mutation introduces new solution elements by occasionally making random changes to the new offspring. For example in Bit Inversion mutation, selected bits are inverted. In the following example, the selected bit is marked in bold: 1**1**001001. After mutation, the new solution is 1**0**001001.

There are various kinds of termination conditions which have been traditionally used in GA. Some of these conditions are described here [SCPB04].

- *Generation Number*: An upper limit on the number of generations can be set by user. The algorithm will terminate when reaching this value.
- *Evolution Time*: A maximum evolution time has to be defined. Whenever the elapsed evolution time exceeds this value, the evolution will be terminated. By default, the evolution is not stopped until the evolution of the current generation has completed, but this behavior can be changed so that the evolution can be stopped within a generation. This is the fair method that is used in this work.
- *Fitness Threshold*: A fitness threshold has to be defined. The algorithm will terminate when the best fitness in the current population becomes less or greater than this value. Being less or greater depend on the definition of the objective value and if the aim is to maximize or minimize the fitness function.
- *Fitness Convergence*: Here the evolution ends when the fitness is deemed as converged. A genetic algorithm is usually said to converge when there is no significant improvement in the values of fitness of the population in the last few generations. However, this convergence should be considered as the convergence of the population to the global optimum. While crossover tries to converge to a specific point in the search area, mutation does its best to avoid convergence and to explore more areas. Whenever a population converges to a local optimum, the algorithm should increase the population diversity to explore other areas. Then it is possible to decide whether the algorithm has reached a convergence of the population to the global optimum.
- *Population Convergence*: The population is deemed as converged when the average fitness across the current population is less than a user-specified percentage away

from the best fitness of the current population. When this criteria is satisfied then the algorithm will be terminated.

- *Gene Convergence*: A gene is deemed as converged when the average value of that gene across all of the chromosomes in the current population is less than a user-specified percentage away from the maximum gene value across the chromosomes. When a user-specified percentage of the genes that make up a chromosome are deemed as converged, the algorithm will be terminated.

The defined GA in this section is very general. There are many things that can and must be implemented differently in various problems. However, GAs tend to thrive in an environment in which there is a very large set of candidate solutions and in which the search space is uneven and has many hills and valleys.

3.2 Local Search

GAs are introduced as a solution to combinatorial optimization problems like the VRPSD. They have been seen as search procedures that can quickly locate high performance regions of vast and complex search spaces. However, it is not always possible to use them for the fine-tuning of those solutions that are very close to optimal ones [GML08].

Hence, GAs may be specifically designed to be mixed with efficient local and neighborhood search techniques to solve NP-hard problems as VRPs [BBC$^+$06]. Local search (LS) can be used on problems that can be formulated as finding a solution maximizing a criterion among a number of candidate solutions [HS04].

Algorithm 3.2: A generic local search in pseudocode

```
1 Input: initial solution $x$
2 Output: possibly improved solution
3 while Termination condition is not satisfied do
4     select $x' \in N(x)$
5     if $x'$ is better that $x$ then
6         $x = x'$
7     end
8 end
```

The LS algorithm starts from an initial candidate solution and explores the search space iteratively within its neighborhood. LS uses a single search path of solutions instead of a search tree. It goes from current solution to a better solution in each iteration. The best solution will be used in the next iteration as current solution until the iteration terminates.

The iteration can be terminated under different criteria. For example the process can terminated based on a time bound or after a given number of steps. Another possible reason for termination could be a continuing process while there is no better solution within the neighborhood of the current solution. Algorithm 3.2 shows a generic local search in pseudocode.

CHAPTER 4

Genetic Algorithm

In this chapter the developed genetic algorithm and its extensions are presented. Section 4.2 and 4.3 describe how the 2-opt local search is combined with the GA and how to benefit from an approximation method, a multi-level evaluation method, in order to get better result. This combination of methods is used to speed up the solution evaluation. The implementation of a complete solution archive is described in section 4.4.

4.1 Structure of the Genetic Algorithm

An overview of GAs is given in section 3.1. This section focuses on the problem specific details of the proposed GA.

Foremost, an initial population is needed which is produced using a stochastic greedy nearest neighbor heuristic algorithm. This algorithm starts at the depot and finds the shortest edges connecting the depot and an unvisited customer. It then sets one of the found customer as a new starting point. This is repeated until all customers have been visited. The sequence of customers will be count as one possible solution. The nearest neighbor algorithm executes quickly but, due to its "greedy" nature, it can sometimes miss shorter routes which are easily noticed with human insight.

The initialization algorithm is described in algorithm 4.1. In the stochastic greedy nearest neighbor heuristic algorithm, the probability of the arc (l, u) from the last customer l added to the tour to any not yet added customer u is $P_U(l, u) = w_{l,u} * \frac{1}{2^{Rank_U(u)}}$. $Rank_U(u)$ is the position of u in a list of all unused U sorted ascending by distances to l and $w_{l,u}$ is the normalized and inverse distance between l and u (the most distant customer has $w = 0$ and the closest $w = 1$). In general, any density function giving a dominating probability to the nearest neighbors can be used instead [Spe13]. Then the next customer is chosen based on these computed probabilities.

Algorithm 4.1: Initialization

```
1 unused = [1, ..., $n$]
2 result = [ 0 ]
3 while $unused.Count > 0$ do
4     // SelectRandom uses PU(l; u)
5     nextCustomer = SelectRandom(result[result.Count-1], unused)
6     result.Add(nextCustomer)
7     unused.Remove(nextCustomer)
8 end
```

To increase the solution quality in a next step, the algorithm performs crossover and mutation operations, which are the two basic operators the GA relies on. As Sperl [Spe13] showed in his work, the custom edge and cyclic recombination operator perform better than a classical edge recombination operator. In this work cyclic recombination is used as well, where a child solution is constructed by taking a random subsequence of one parent and then following the other parent from the end of the random subsequence, omitting already visited customers.

The mutation operator is selected randomly among the following three operators:

- Swap mutation where two customers change their position in the tour.
- Subsequence reverse mutation where a random subsequence of the tour is reversed.
- Weighted insert mutation where a random customer at position c is inserted at position p in tour s with probability $P_c(p) = \frac{1}{2^{Rank_c(p)}}$ where $Rank_c(p)$ is the position of p in a list of customers sorted by $d(c,p) = d_{s(c-1),s(c+1)} - d_{s(c-1),s(c)} - d_{s(c),s(c+1)} - d_{s(p-1),s(p)} + d_{s(p-1),s(c)} + d_{s(c),s(p)}$. The intention is that a random customer is removed and inserted more likely into a part of the tour that passes close by.

4.2 Local Search

One of the powerful local search methods that can be combined with GA for optimizing VRPs is 2-opt. The 2-opt neighborhood structure was first proposed by Croes in 1958 for solving the traveling salesman problem [Cro58]. It works on a tour and removes two edges in a tour, and then one of the resulting segments is reversed and the two segments are reconnected. Once the two edges to be deleted have been chosen, there is no choice about which edges to add since there is only one way to add new edges that results in a valid tour.

If 2-opt results in an improved tour, the change is preserved. Otherwise, the tour is returned to the original form. Compared with the 2-opt, k-Opt ($k = 3, 4, \dots$) uses more edges to rearrange the tours [GITN04].

In the first step, the 2-opt is added to the GA. As described below in algorithm, 4.2 line number 7 is added to the standard GA. Applying 2-opt to the GA improves convergence and provides the GA with a search intensification. Applying 2-opt to every single individual reduces the performance, hence 2-opt is applied to only a fraction of the individuals of the whole population.

Algorithm 4.2: Optimized GA using 2-opt

```
1 Initialize randomly initialize population (t)
2 Evaluate determine fitness of population (t)
3 while Termination condition is not satisfied do
4     Select select parents from population (t)
5     Crossover perform crossover on parents creating population (t+1)
6     Mutate perform mutation of population (t+1)
7     2-opt optimization apply 2-opt to population (t+1)
8     Evaluate determine fitness of population (t+1)
9 end
```

Figure 4.1 shows one step of the 2-opt local search. The left side of the figure shows an initial route and an improved version found on the right side. The 2 - 3 and 4 - 5 links on the left have been removed and replaced by the 2 - 4 and 3 - 5 edges in red on the right. Notice also that the path 2 - 3 - 4 - 5 on the left was reversed to 2 - 4 - 3 - 5 on the right to keep the route valid.

After moving to a neighbor solution in the 2-opt procedure, the improvement is calculated which can be done in at least two reasonable ways. First, the improvement can be approximated by computing only the distance as in a normal VRP problem. Assume we remove the edges from node A to $A1$ and from node B to node $B1$. Then the improvement is computed as follows:

$$\begin{aligned}\text{temp-improvement} = \text{distance from } nodeA \text{ to } nodeB - \\ \text{distance from } nodeA \text{ to } nodeA1 + \\ \text{distance from } nodeA1 \text{ to } nodeB1 - \\ \text{distance from } nodeB \text{ to } nodeB1\end{aligned} \tag{4.1}$$

However, this approximation does not take the randomness of the demand into account. So another possibility is to compute the exact objective value of the neighbor solution using the dynamic programming algorithm.

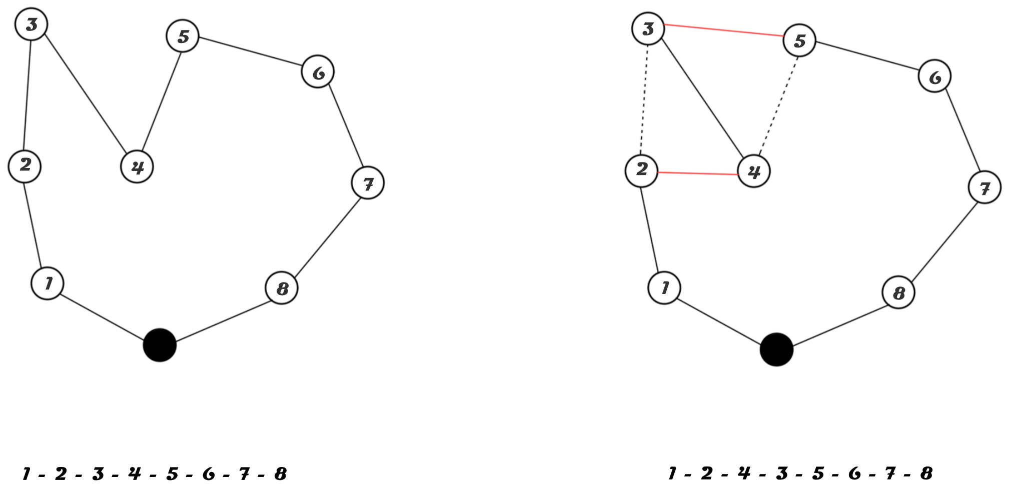


Figure 4.1: 2-opt move

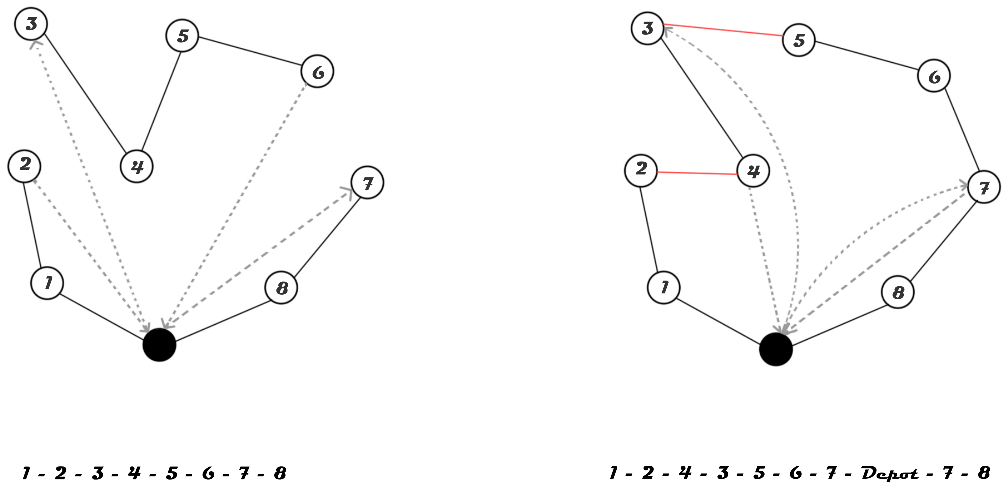


Figure 4.2: 2-opt considering exact evaluation parameters

Figure 4.2 shows the same route as depicted in figure 4.1 but additionally considers the criteria of the VRPSD before and after applying 2-opt procedure illustrated by the dashed lines representing restocking trips.

Algorithm 4.3: Exact 2-opt

```
1  obj = objective of currentRoute
2  bestObj = obj
3  while Termination condition is not satisfied do
4      for $i \leftarrow 0$ to $n$ do
5          nodeA = customer $i$ in currentRoute
6          for $j \leftarrow i + 1$ to $n - 1$ do
7              nodeB = customer $j$ in currentRoute
8              newRoute = invert node order between nodeA and nodeB
9              temp-obj = compute objective value of newRoute
10             if temp-obj < bestobj then
11                 bestobj = temp-obj
12                 temp-route = newRoute
13             end
14         end
15     end
16     if bestObj < obj then
17         currentRoute = temp-route
18         obj = bestObj
19     end
20 end
```

Algorithm 4.3 shows the complete 2-opt local search. There it can be seen that the algorithm iterates over the tour and makes all possible 2-opt moves. After each move, in line 11, the exact fitness of the neighbor solution is calculated. Then the best neighbor is chosen as next starting solution. The process of calculating the exact value via the dynamic programming algorithm is time consuming. In the next section, it will be described how the multi-level evaluation procedure is used to speed up this process.

4.2.1 Dynamic Programming

Dynamic programming (DP) is a popular algorithm design technique which is used to solve combinatorial search and optimization problems. A DP algorithm stores and examines the previously solved sub problems and combines their solutions to give the best solution for the given problem [Bel54]. In DP, a single step is sequentially related to the preceding steps and is not itself a solution to the problem. A single step contains information that identifies a segment or a part of the optimal solution. In DP, the time is usually reduced by increasing the amount of memory/space needed.

For solving a problem using DP, the problem should have an optimal substructure and overlapping subproblems. Having an optimal substructure means that the solution to a given optimization problem can be obtained by the combination of optimal solutions to its subproblems. Consequently, the first step towards devising a dynamic programming solution is to check whether the problem exhibits such optimal substructure[BBBB95].

Overlapping subproblems mean that the space of subproblems must be small, that is, any recursive algorithm solving the problem should solve the same subproblems over and over rather than generating new subproblems [BBBB95].

In this study, a GA is used to generate the sequence of customers and each sequence of customers is evaluated using a dynamic programming algorithm. Equation 1.1 shows the dynamic programming formulation to calculate the total expected costs of a sequence. The implementation of this formulation is shown in algorithm 4.4, which runs in $O\left(nQ^2\right)$ time. The memory requirement is $O\left(nQ\right)$ if one stores all intermediate values for $f_j\left(q\right)$, and $O\left(Q\right)$ otherwise.

Algorithm 4.4: Objective function (fitness) of an a priori tour

```
1  $Q$ = maximum vehicle capacity
2  $q$ = remaining load of vehicle
3  $j$ = visisted customer
4  $h_n$ = threshold
5  for $q \in (Q, Q-1, ..., 0)$ do
6      $f_n\left(q\right)$ = the travel costs (distances) between depot and customer $n$
7      for $j = (n-1, n-2, ..., 1)$ do
8          $f_j^r\left(q\right)$ = the expected cost when replacing the direct route with a preventive
           restock (see 1.3)
9          $h_n = Q$
10         for $q = (Q, Q-1, ..., 0)$ do
11             $f_j^p\left(q\right)$ = the expected cost when proceeding directly to the next
               customer (see 1.2)
12             $f_j(q) = min\left\{f_j^p(q), f_j^r(q)\right\}$
13             if $f_j^p\left(q\right) \geq f_j^r\left(q\right)$ then
14                 $h_n = q$
15             end
16         end
17     end
18 end
19 output $f_n\left(Q\right)$
```

4.3 Multi-level evaluation scheme

In addition to the DP approach, a multi-level evaluation scheme (ML-ES) is also proposed for faster solution evaluations. ML-ES iteratively approximates the quality of a solution candidate with increasing accuracy until it can be discarded or it is evaluated exactly. In the deterministic case of the VRPSD, each customer's demand is defined clearly but in the stochastic case, this demand is a stochastic variable and only the probability of each demand value is given.

Here, the focus is on creating a simple multi-level analysis model to avoid unnecessary calculations. The preliminary tests showed that the best results were obtained at the first three levels. Hence, the objective value is calculated at three levels. On each level, a new objective value will be calculated for the same route but for a the vehicle with a lower capacity. As a new capacity is applied to the vehicle, the probability of customer's demands should be updated as well.

On the first level, the new capacity will be calculated by dividing the original capacity by two. When the fitness value of the route (computed using the DP algorithm) with a vehicle of reduced capacity is worse than or equal to the fitness value of the best found route so far, it will not be necessary to evaluate this solution candidate more accurately because, as we will see later, its exact objective value cannot be better than the current best solution.

If the fitness of the route with new vehicle capacity is better than the best fitness, the procedure continues until the next level is reached. The same routine will be performed on the second level. Here, the capacity will be divided by four to calculate the new capacity, while it will be divided by eight on the third level. If the process finds a better fitness on all three levels, it can be true that the new route has a better fitness value than the old one.

Algorithm 4.5 shows the exact steps taken using DP and ML-ES.

Algorithm 4.5: Exact fitness evaluation using multi-level evaluation scheme

```
1 numberOfLevel = 3
2 while newFitness < bestFitness and numberOfLevel != 0 do
3     newFitness = compute fitness according to numberOfLevel
4     numberOfLevel = numberOfLevel -1
5 end
6 return newFitness
```

Applying this three level approach, the performance of the overall algorithm improves efficiently as can be seen in the computational results in chapter 5.

Next, we will show why the solution evaluation using this method is still exact for solutions that could improve the best found solution. By scaling down the capacity and adapting the probability distributions, the resulting value on each level is a lower bound to the objective value of the upper level. This can be proved as follows.

In the ML-ES, which is applied to GA in this work, there are 3 levels of approximation, where level 0 is the exact evaluation and third level is the roughest approximation level. Starting with level 0, increasing the level by one means to scale down the vehicle capacity Q and all demand distributions p_{jk} by a factor of two. Here a new vehicle capacity Q^i will be introduced and new probabilities p^i_{jk} subject to level i, which are defined in the following way [BHR15]:

$$Q^0 = Q \tag{4.2}$$

$$p^0_{jk} = p_{jk}, \forall j = 1, ..., n, k = 0, ..., Q^i \tag{4.3}$$

$$Q^i = \left\lceil \frac{Q^{i-1}}{2} \right\rceil, \forall i = 1, ..., 3 \tag{4.4}$$

$$p^i_{jk} = p^{i-1}_{j,2k} + p^{i-1}_{j,2k+1}, \forall j = 1, ..., n, k = 0, ..., Q^i, \forall i = 1, ..., 3 \tag{4.5}$$

Figure 4.3 shows exemplarily for one node how the probability distribution for the demand changes at each level [BHR15].

Not only is level $i \geq 1$ an approximation, but its objective value is also a lower bound for the objective value of the preceding level $i-1$, which will be showed next.

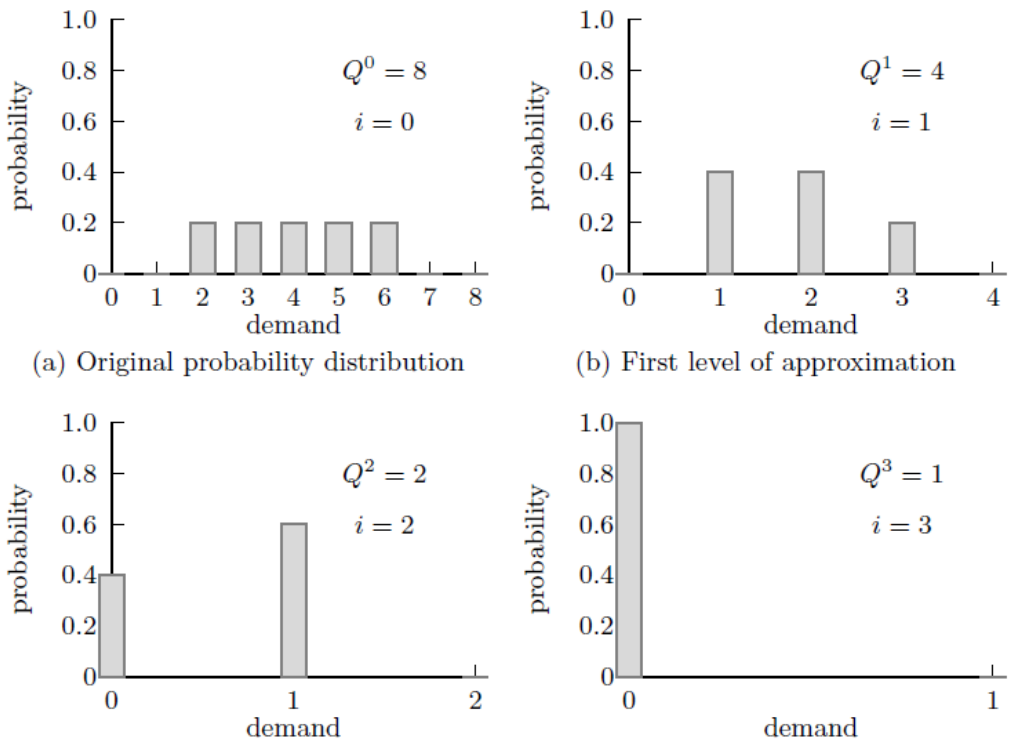
Lemma 1. *With increasing level i the ratio of the scaled expected demand of each node to the vehicle capacity Q^i is non-increasing [BHR15].*

Proof. It hast to be shown that for each node j and each demand $0 \leq k \leq Q$ that

$$\frac{\sum_{k=0}^{Q^i} kp^i_{jk}}{Q^i} \leq \frac{\sum_{k=0}^{Q^{i-1}} kp^{i-1}_{jk}}{Q^{i-1}} Q^{i-1}, \forall i = 1, ..., 3$$

is valid. Suppose to the contrary that for one node j and one demand k the following holds:

$$\frac{kp^i_{jk}}{Q^i} = \frac{k\left(p^{i-1}_{j,2k} + p^{i-1}_{j,2k+1}\right)}{\frac{Q^{i-1}}{2}} > \frac{2kp^{i-1}_{j,2k} + (2k+1)\, p^{i-1}_{j,2k+1}}{Q^{i-1}} = \frac{kp^{i-1}_{jk}}{Q^{i-1}}$$



(a) Original probability distribution (b) First level of approximation

(c) Second level of approximation (d) Highest level of approximation

Figure 4.3: An exemplary demand probability distribution and its different levels of approximation [BHR15].

$$2kp_{j,2k}^{i-1} + 2kp_{j,2k+1}^{i-1} > 2kp_{j,2k}^{i-1} + 2kp_{j,2k+1}^{i-1} + p_{j,2k+1}^{i-1}$$

$$0 > p_{j,2k+1}^{i-1}$$

Obviously, this is a contradiction because all probabilities must be non-negative. Therefore, as no $\frac{kp_{jk}^{i}}{Q^{i}}$ can be larger than $\frac{kp_{jk}^{i-1}}{Q^{i-1}}$ for any node j this also holds for the sum over all demands, which proves the Lemma [BHR15].

Theorem 1. *Let $c^{i}(t)$ be the objective value of a tour t on approximation level i. For each tour t it holds that $c^{i}(t) \leq c^{i-1}(t), \forall i = 1, ..., 3$ [BHR15].*

Proof. Due to Lemma 1 it follows that the total expected relative demand of all nodes on level i is smaller or equal to that of level $i-1$. So we can possibly service more customers before a restocking is needed and therefore the resulting objective $c^i(t)$ value is a lower bound to the exact objective value $c^0(t)$ and to the objective value at the preceding level $c^{i-1}(t)$ [BHR15].

As it has been observed in the computational results, applying these technique improved the result significantly. In next section, it has been tried to apply a memory-efficient structure for keeping track of already evaluated solution candidates.

4.4 Solution Archive

There are several ways to classify metaheuristic algorithms. One way is to classify them based on the usage of the memory. The metaheuristics can be divided into memory usage algorithms such as Tabu search and memoryless algorithms such as simulated annealing [BK78].

In genetic algorithms, identical solutions may be generated repeatedly because GAs do not keep track of the history and already evaluated solutions are often revisited. What is more, there is no assurance that a GA will find a global optimum and that the algorithm will not start to generate duplicate solution candidates.

One way to speed up the algorithm and possibly avoiding duplicate solutions is to avoid visiting a solution repeatedly. There exist various kinds of population management strategies to counteract premature convergence and to ensure a reasonable diversity in the population.

In this work, a complete solution archive proposed by Raidl and Hu [RH10] is implemented. The complete solution archive is based on a memory efficient trie data structure. It is implemented in order to (a) efficiently detect already evaluated candidate solutions and to (b) efficiently transform them by applying typically small adaptations into yet unconsidered candidate solutions [RH10].

A trie is an efficient information retrieval data structure that is used to store a dynamic set or associative array where the keys are usually strings. The main use of trie is for large dictionaries of english words in spell-checking and natural language understanding programs. The idea is that all strings which have a common stem or prefix hang off a common node. The elements in a string can be recovered by scanning from the root to the leaf that ends the string. All strings in a trie can be recovered by doing a depth first scan of the tree.

Here, a trie is used to save all generated solutions. Every node of the trie consists of multiple sequence of nodes and each branch represents a possible solution candidate. Each entry of each node of the trie can obtain one of the following three states at a time. It can contain an object which is either a pointer to the rest of a possible solution or a null object, or as another state signed by 'C' as a complete flag to show a leaf node.

Inserting a customer as a key into the trie is a simple approach. Every customer of an input solution candidate is inserted as an individual trie node. The children are an array of pointers to next level trie nodes. The key acts as an index into the array children.

The insertion procedure is described in algorithm 4.6

Algorithm 4.6: Inserting a solution candidate into the trie

```
Candidate solution $(x_1, ..., x_n)$
q = root
l = 1
while q != 'C' and l != n do
    if (q.next($x_l$) == null) then
        q.next($x_l$) = new trie node
    end
    if (q.next($x_l$) == 'C') then
        return true
    end
    q = q.next($x_l$)
    l = l + 1
end
q.next($x_l$) = 'C'
prune the trie bottom up
return false
```

To have a better understanding of the insertion of a solution into a trie, take the example below using a solution found for delivering to four customers:

[4, 3, 2, 1], [4, 3, 1, 2]

As can be seen in figure 4.4 (1) the first solution is inserted into an empty trie and the value of each element of trie is set to null. In the next step, the rest of the solution has to be inserted, which started with customer number three. The customer three has to be found on the first level of the trie while the rest of the solution is inserted as a new child for this customer. The value of that element will be replaced with a pointer to the rest of the solution. This is depicted in figure 4.4 (2).

This will be repeated until the last customer will be inserted as a leaf node and the value of the node set to 'c' to show that this branch is completed (figure 4.4 (4)). The same steps have to be taken for inserting the second solution which can be seen in figure 4.4 (5).

Using trie data structure, search complexities can be brought to optimal limit which is equal to the length of the solution (L). Using a trie, the search process of a key can be accomplished in O(L) time. However, trie storage requirements are another important

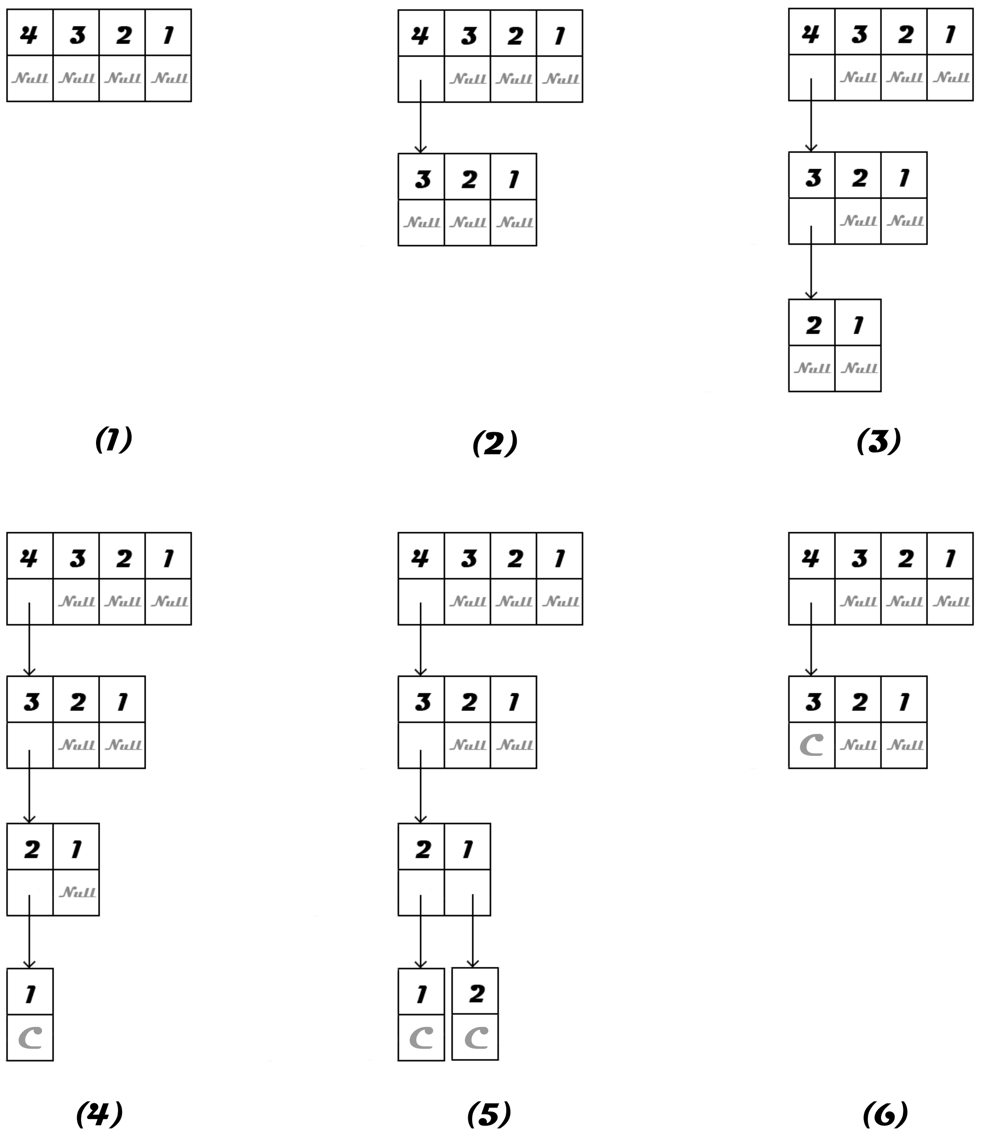


Figure 4.4: Inserting a solution into an empty trie data structure

factor that should be considered. Inserting found solutions one by one enlarges the trie over the time. It is also very important to keep the tree as small as possible to reduce the search effort. For this aim, a technique should be used for pruning the trie. Pruning can occur in a top down or bottom up fashion. A top down pruning will traverse nodes and trim subtrees starting at the root, while a bottom up pruning will start at the leaf nodes.

In this thesis, bottom up pruning is applied. For this reason, the trie structure will be checked after inserting a new solution. If all leaf nodes of a subtree have the 'C', all the leaves will be cut and the parent flag set to 'C', which was previously set with the pointer to the rest of the found solution. This process is depicted in figure 4.4 (6).

The process of pruning the trie can continue until the root of the trie can be set to 'C'. This indicates that all solutions of the whole search space have been added, and the GA can be terminated returning its best found solution as (proven) optimum. However, this might happen in rare cases only.

In many problems, GAs may have a tendency to converge towards local optima or even arbitrary points rather than the global optimum of the problem. A local optimum of an optimization problem is a solution that is optimal (either maximal or minimal) within a neighboring set of candidate solutions. This is in contrast to a global optimum, which is the optimal solution among all possible solutions, not just those in a particular neighborhood of values.

To help the GA to find the best solution, another method provided inside the trie is responsible for generating new solutions based on not yet visited paths in trie data structure. For this aim, every time that the insertion finds a duplicate it automatically generates a new solution, inserts it into trie and returns it back to the GA for calculating the fitness. Algorithm 4.7 shows how the algorithms converts an existing solution.

The entire process of how the solution archive works is summarized in 4.8. The described steps in algorithm 4.8 should be integrated into the GA. This part is placed after '2-opt optimization' step and should be applied to each generated individual.

Algorithm 4.7: Convert a duplicate solution

```
1  **Input** x, nodes
2  q = random node from nodes
3  l = level of q
4  $x_l^{'}$ = value of randomly chosen non-complete entry of q
5  swap $x_l$ with $x_l^{'}$
6  **while** *q != 'C'* **do**
7      **if** *q.next($x_l$) == null* **then**
8          q.next($x_l$) = new trie node
9      **end**
10     **if** *q.next($x_l$) == 'C'* **then**
11         q.next($x_l$) = new trie node
12         $x_l^{''}$ = value of randomly chosen non-complete entry of q
13         swap $x_l^{'}$ with $x_l^{''}$
14         continue
15     **end**
16     q = q.next($x_l$)
17     $l = l + 1$
18 **end**
19 q.next($x_l$) = 'C'
20 prune the trie bottom up
21 return x
```

Algorithm 4.8: High-level description of the solution archive

```
1  solutionArchive = an instance of solutionArchive
2  Solution = An individual which is generated using GA
3  bool solutionExisted = solutionArchive.AddNewSolution(Solution)
4  **if** *solutionExisted==true* **then**
5      newPath = solutionArchive.convertSolution
6  **end**
```

CHAPTER 5

Computational Results

An overview of the problem and a comprehensive description of the solutions are given in chapters 1 and 4. A valid approach to evaluate the discussed algorithm is to run computational tests, whose construction is described in section 5.1 of this chapter. Based on the data resulting from the tests, the different results will then be further investigated in section 5.2.

5.1 Construction of Test

The tests for the VRPSD were designed on the basis of an existing algorithm from [Spe13] and instances from Bianchi[1]. These instances are interesting since the position of customers was not chosen uniformly at random but randomly with normal distributions around two centers (so customers are grouped in two clusters). This is done in order to consider instances that reflect real world situations, where customers may, for instance, be located in two different cities. The clusters' centers have coordinates in [0,99], and customers' coordinates are all different.

Fifteen random instances have been chosen with 50 customers. Two main strategies have been chosen to test the instances and evaluate the results. The first strategy focuses on finding a good population size to produce good results within a limited timeframe. The second strategy consists of different optimization methods being applied to the default genetic algorithm. This default genetic algorithm, in turn, is already applied to the optimal population size, which has been determined based on the results of the first strategy.

In the first strategy the basic algorithm is used to find an optimal population size. Testing different methods in the second strategy has been performed based on the result in this

[1]`http://iridia.ulb.ac.be/supp/IridiaSupp2004-001/index.html`

step. Here the algorithm has been run for population sizes of 100, 300, and 500. Then in the second strategy computational experimental tests have been run after applying different optimization methods: Local search, multi-level evaluation scheme, and the solution archive. These will be discussed in detail later in this chapter.

Before going into greater detail, several terms in in the table of results should be described.

"Objective value" is the average value calculated by the algorithms over all (here 30) runs and is shown as "Obj. Val." in tables. **"Standard deviation"** is a measure that is used to quantify the amount of variation or dispersion of a set of data values. A standard deviation close to 0 indicates that the data points tend to be very close to Objective mean. These values are calculated for each instance. To have a good overview of all instances, **"Objective mean"** and **"Objective geometric mean"** also calculated separately to give an overview of all runs over all instances at once.

"Objective mean" shows the average value over all runs and over all instances and can be seen as "obj. mean" in the tables. **"Objective geometric mean"** is a type of average, which indicates the central tendency of a set of value by using the product of these values, as opposed to the objective mean, which uses their sum. In the tables "obj. g. mean" refers to this term.

Furthermore, the Wilcoxon rank-sum test, that compares results to the objective values, will be run to better highlight the improvements. This test is a nonparametric test which can often be used provided the two independent samples are drawn from populations with an ordinal distribution. This test is also applied to the results to find a significant difference between configurations.

5.2 Computational Results

All tests were run on an Intel Core i3 1.90GHz computer with 4 GB of RAM. Evolution time has been selected as the termination condition and the maximum evolution time has been set to 10 minutes. All reported statistics are averages of over 30 runs per instance.

5.2.1 Test on population size

Attempting to solve optimization problems using GAs raises the problem of finding a good set of solutions and finding a good initial population. Moreover, the probability and type of crossover, the probability and type of mutation, the stopping criteria, the type of selection operator, and the fitness function play an important role in finding a good solution and in the performance of GA.

The results of the computational tests performed on most of the named parameters are published in [Spe13]. Here, the focus lies on finding a good initial population. Table 5.1

shows the configured value of the used parameter for running the computational tests. These parameters are as follows:

- **Mutation Rate** is the probability that the new solution is mutated.
- **Population Size** is number of population in each generation of GA
- **Survival Rate** specifies a percentage of the fittest solutions which are used exclusively to generate the next generation.
- **Tournament Size** is the size of the tournament selector. It is a method of selecting an individual from a population of individuals in a genetic algorithm.
- **Number of Runs per Problem Instance** which shows how many times the GA runs for one instance of problem

Table 5.1: Used parameter for capturing the result of computational tests on GA

Parameter	Value
Mutation Rate	0.25
Population Size	500
Survival Rate	35
Tournament Size	6
Number of Runs per Problem Instance	30

In GAs, each candidate solution is an individual in a population. Research [KK06] shows that population size plays an important role in evolutionary computation like GAs. Hence, in the first step, an attempt was made to find a reasonably good population size. The determined population size will be used later for designing a computational experimental analysis based on different optimization techniques.

The results in table 5.2 show that small population sizes render the algorithm unreliable, and large population sizes affect the computation time in finding a solution. Due to the significant influence of population size on the solution quality and search time, finding a trade-off between population size and computation time is necessary to obtain a good final within a reasonable timeframe.

Table 5.2 gives an overview of the result of the GA for population sizes of 100, 300, and 500 on 15 different instances. These population sizes are selected from many other tested population sizes and the most stable results, based on termination condition, are selected as the experiment's results.

A higher level overview over all runs over all instances is given in the last row of the table, where the objective mean and the objective geometric mean for the different population

Table 5.2: Computational results on instances using default GA for population size of 100, 300, and 500

Instance	100 Population		300 Population		500 Population	
	Obj. Val.	Std. Dev.	Obj. Val.	Std. Dev.	Obj. Val.	Std. Dev.
s05551	1576.3	10.4	**1560.5**	14.3	1563.3	9.0
s55510	2734.6	16.5	2721.0	12.3	**2715.8**	14.7
s55515	1875.1	9.1	1868.9	7.9	**1856.8**	9.7
s55520	1858.5	14.8	1839.2	9.7	**1838.2**	10.2
s55525	1718.2	12.0	1703.1	13.7	**1705.0**	12.5
s55530	2234.8	15.7	2223.1	12.4	**2211.9**	10.1
s55535	2074.6	9.6	2064.0	11.0	**2059.8**	8.1
s55540	1954.9	10.5	1947.7	8.9	**1947.1**	9.5
s55545	1455.1	15.0	1435.8	12.2	**1435.6**	9.4
s55550	2725.6	13.3	2711.1	19.3	**2705.8**	15.4
s55555	2014.0	13.4	2003.5	12.2	**2002.3**	12.3
s55560	2612.8	14.0	2599.6	13.6	**2589.9**	10.6
s55565	2799.4	15.1	2785.4	12.4	**2776.9**	12.4
s55570	2071.7	11.2	2063.1	9.6	**2056.3**	8.8
s55575	2474.8	13.3	2473.2	7.1	**2468.0**	10.0

sizes are shown. It has been recognized that if the initial population increases from 100 up to 300, the objectives mean decreases from 1977.3 down to 1971.7 and when initial population increases from 300 up to 500, the objectives mean decreases from 1971.7 down to 1968.7.

The objective mean and the objective geometric mean over all runs are also summarized in table 5.3.

Table 5.3: Objective mean and the objective geometric mean of different population size

Population	100	300	500
obj. mean	2145.3	2133.3	2128.8
obj. g. mean	2103.9	2091.3	2087.3

The summary of statistical results can be taken from table 5.4. The header of rows and columns shows the population size and the cells contains the frequency of better results in percentage. The statistics parameters, which are used to obtain these results, are the calculated objective values in table 5.2.

Observing every single instance has shown that as the size of population increased from 100 to 300 or from 100 to 500, the objective values have improved in all instances. It was

Table 5.4: Statistical results of increasing population size

Population	100	300	500
100	—	100%	100%
300	0%	—	81.5%
500	0%	12.5%	—

also expected that increasing the population from 300 to 500 would lead to better results. However, not all but only 81.5% (table 5.4) of instances have shown improvement.

Table 5.5: Wilcoxon rank-sum results on population size

Population	100	300	500
100	-	0	0
300	14	-	0
500	15	7	-

The results of running the Wilcoxon rank-sum test is summarized in table 5.5. It shows that the small population sizes are never better than the bigger population sizes. The population size of 300 leads to better results in 14 instances in comparison to the population size of 100 and the population size of 500 is better than the population size of 100 in all 15 instances. However, comparing the population size of 500 to the population size of 300, the results are better only in 7 instances.

5.2.2 Applying optimization method

Taking into account the results of section 5.2.1, it is reasonable to investigate how the optimization method affects the GA in generating a better solution for an initial population size of 500. As described in chapter 4, mainly two methods are used to optimize the performance of the GA. Firstly, the multi-level evaluation scheme and secondly, the complete solution archive are implemented.

The exact evaluation of the solution based on dynamic programming was very time consuming. Therefore, a multi-level evaluation was applied, which significantly reduced the evaluation time of the candidate solution.

In order to verify the advantages of each method separately, the tests are run once after the application of the multi-level evaluation scheme, and then additionally after implementing the complete solution archive for a population size of 500.

Table 5.6 allows for a comparison of the results after applying the optimization methods

Table 5.6: Comparison of applying optimization method on the population size of 500

Instance	Default GA		ML-ES		ML-ES + Sol. Arch.	
	Obj. Val.	Std. Dev.	Obj. Val.	Std. Dev.	Obj. Val.	Std. Dev.
s05551	1563.3	9.0	1416.8	15.7	**1410.5**	15.3
s55510	2715.8	14.7	**2534.9**	12.1	2540.9	12.6
s55515	1856.8	9.7	1724.3	9.8	**1722.0**	12.9
s55520	1838.2	10.2	**1680.5**	14.1	1688.7	13.4
s55525	1705.0	12.5	**1570.7**	9.3	1576.8	13.7
s55530	2211.9	10.1	2077.2	11.5	**2068.7**	15.1
s55535	2059.8	8.1	1879.9	12.7	**1877.1**	15.6
s55540	1947.1	9.5	**1797.4**	16.6	1802.8	10.7
s55545	1435.6	9.4	**1320.8**	9.6	1322.3	18.2
s55550	2705.8	15.4	2525.2	16.0	**2522.8**	14.1
s55555	2002.3	12.3	**1830.8**	15.5	1833.5	18.5
s55560	2589.9	10.6	2379.8	18.3	**2376.3**	18.4
s55565	2776.9	12.4	**2572.0**	14.8	2583.0	21.8
s55570	2056.3	8.8	**1916.2**	10.4	1916.8	11.5
s55575	2468.0	10.0	**2303.7**	14.3	2305.5	18.4

on the default GA. The first column shows the instances, columns 2 and 3, respectively, contain the objective value and standard deviation of the default GA while columns 4 and 5 show the values after applying the multi-level evaluation scheme. Finally, columns 6 and 7 give the results after the complete solution archive has been applied.

It is also important to note that ML-ES is able to reduce the overall runtime of the algorithm, which allows for setting a timeframe of 10 minutes as termination condition. The algorithm was able to generate and evaluate more solution candidates, which increases the reliability of the final solution.

The application of the multi-level evaluation scheme improved the results significantly. In this case, the objective mean decreases from 2128.8 to 1968.7. It can also be observed that this method leads to an improvement in all instances while applying the solution archive fails to result in significant improvements over most instances. However, in comparison to the default GA, a significant improvement can be reached by using the solution archive. In the latter case, the objective mean decreases from 2128.8 to 1969.9.

Table 5.7 gives a better overview of applying optimization methods on the population size of 500.

The results of running the Wilcoxon rank-sum test show that applying ML-ES leads to better results in all 450 runs in comparison to the default GA. However, comparing the results using the ML-ES method with and without applying the solution archive, fails to

Table 5.7: Objective mean and the objective geometric mean of all runs after applying optimization method on the population size 500

Population	Default GA	ML-ES	ML-ES + Sol. Arch.
obj. mean	2128.8	1968.7	1969.9
obj. g. mean	2087.3	1928.1	1929.2

show a clear improvement due to the solution archive. Out of 450 runs, ML-ES without applying the solution archive was better than ML-ES combined with the archive in 235 runs.

The results of the Wilcoxon rank-sum test on different optimization methods are summarized in table 5.8. The data shows that applying optimization methods has improved the default GA significantly. The ML-ES was better than ML-ES using the solution archive in 3 instances while ML-ES combined with the solution archive was better than ML-ES only in 2 instances.

Table 5.8: Wilcoxon rank-sum test results on applying optimization method

Population	Default GA	ML-ES	ML-ES + Sol. Arch.
Default GA	-	0	0
ML-ES	15	-	3
ML-ES + Sol. Arch.	15	2	-

5.2.3 Results of applying ML-ES and DP

Additionally, table 5.9 and table 5.10 show the intermediate results of applying ML-ES. These results are only examples and cannot be generalized, partly because they were gathered after different runs. The solutions differ from run to run and also the duration, shown in table 5.9 and table 5.10, may vary from run to run for the same instance.

The table 5.9 shows the fitness value before (Main Fitness) and after (Best Fitness) applying the ML-ES. The third column shows the duration of calculating the Best Fitness using ML-ES.

Table 5.10 shows the calculated objective value and the duration of this calculation on three different levels of ML-ES.

Taking instance s05551 as an example, it can be taken from table 5.9 that the best fitness was calculated within 00:00:26.2 seconds, which is fairly close to the duration of the same instance on the first level in the table 5.10. This means that the best fitness of this

Table 5.9: Intermediate result of testing ML-ES

	Main Fitness	Best Fitness	Duration
s05551	1662.2	1519.9	00:00:26.2
s05552	2434.4	2222.4	00:00:29.2
s05553	1539.7	1392.5	00:00:11.6
s05554	1674.9	1544.6	00:00:22.1
s05555	2387.0	2183.7	00:00:17.0
s05556	2562.2	2450.6	00:00:17.5
s05557	2626.4	2455.0	00:00:20.0

instance is calculated on the first level of the algorithm. However, looking at instance s05553, it becomes clear that this value was calculated on the second level of the ML-ES, because the registered duration in table 5.9 is close to the calculated duration of the second level.

Table 5.10: Intermediate result of testing ML-ES

	level 1		level 2		level 3	
	Obj. Val.	Duration	Obj. Val.	Duration	Obj. Val.	Duration
s05551	1609.3	00:00:28.2	1592.6	00:00:09.3	1496.1	00:00:03.5
s05552	2214.1	00:00:33.0	2172.0	00:00:10.3	2074.0	00:00:04.1
s05553	1375.1	00:00:39.3	1364.6	00:00:11.4	1343.4	00:00:04.2
s05554	1474.1	00:00:37.0	1453.4	00:00:11.2	1402.8	00:00:04.2
s05555	2146.0	00:00:31.5	2016.5	00:00:09.7	1957.1	00:00:03.8
s05556	2512.5	00:00:36.2	2445.9	00:00:10.8	2380.2	00:00:04.1
s05557	2458.7	00:00:39.2	2393.4	00:00:12.1	2309.8	00:00:04.3

5.2.4 Comparison with existing approaches

The main goals of the two computational experiments in section 5.2.1 and 5.2.2 are:

- The analysis of the GA performance for different population sizes. The results show that the default GA performs better with a population size of 500.
- The analysis of the existing GA performance after applying the optimization method.

Two optimization methods were applied, namely multi-level evaluation and the complete solution archive, both of which lead to significant improvements in the results.

In this section, the best result of section 5.2.2 is compared with Genetic Algorithm (GA) and Iterated Local Search (ILS). The latter was described by Bianchi et al. [BBC$^+$06]. Bianchi et al. [BBC$^+$06] analyzes the performance of metaheuristics on the vehicle routing problem with stochastic demands (VRPSD).

Since the performance comparison between applied optimization methods shows that multi-level evaluation leads to a better result than the complete solution archive, the results of the multi-level evaluation optimization method have been used for the comparison.

Table 5.11: Comparison of applying ML-ES with existing approaches

Instance	Multi-level evl.		Bianchi (GA)		Bianchi (ILS)	
	Obj. Val.	Std. Dev.	Obj. Val.	Std. Dev.	Obj. Val.	Std. Dev.
s05551	**1416.8**	15.7	1464.6	7.9	1458.3	5.2
s55510	**2534.9**	12.1	2596.1	10.3	2590.9	9.2
s55515	**1724.3**	9.8	1792.2	5.1	1792.1	5.3
s55520	**1680.5**	14.1	1749.9	7.0	1744.1	11.4
s55525	**1570.7**	9.3	1616.9	6.5	1621.3	5.5
s55530	**2077.2**	11.5	2119.2	7.1	2107.4	10.9
s55535	**1879.9**	12.7	1977.6	7.3	1975.1	5.8
s55540	**1797.4**	16.6	1858.7	3.7	1859.3	8.1
s55545	**1320.8**	9.6	1339.6	5.4	1336.4	7.4
s55550	**2525.2**	16.0	2575.0	5.2	2564.4	6.5
s55555	**1830.8**	15.5	1909.0	10.8	1907.2	7.3
s55560	**2379.8**	18.3	2487.2	6.4	2484.2	3.5
s55565	**2572.0**	14.8	2671.6	7.6	2666.8	6.0
s55570	**1916.2**	10.4	1972.0	7.2	1961.1	8.0
s55575	**2303.7**	14.3	2396.8	3.5	2389.4	3.7

To make the comparison as accurate as possible, all the tests should be run under the same conditions. Therefore, the two algorithms, which are taken from Bianchi `http://iridia.ulb.ac.be/supp/IridiaSupp2004-001/index.html`, are run on the same machine. It is also worth mentioning that two versions of metaheuristic exist based on the type of approximation scheme used in the local search: VRPSD-approximation or TSP-approximation. In the latter case, the algorithms utilize TSP-approximation, as the authors showed that it performed better.

The results are shown in table 5.11. Bianchi (ILS) with an objective mean of 2030.5 is better than Bianchi (GA) with an objective mean of 2035.1 in 81.25% of the instances. It is evident that the multi-level evaluation with a mean objective of 1968.7 is better than both Bianchi (ILS) and Bianchi (GA), and significantly improved the final objective value.

A summary of the objective mean and the objective geometric mean of applying ML-ES to default GA and existing approaches can be found in table 5.12.

Table 5.12: Objective mean and the objective geometric mean of applying ML-ES with existing approaches

Population	Multi-level evl.	Bianchi (GA)	Bianchi (ILS) Arch.
obj. mean	1968.7	2035.1	2030.5
obj. g. mean	1928.1	1992.9	1988.6

CHAPTER 6

Conclusions and Future Work

The Vehicle Routing Problem is of central importance in distribution management. There exist several versions of the problem, and a wide variety of exact and approximate algorithms have been proposed for solving them. This work considers a metaheuristic based on an evolutionary algorithm for solving the Vehicle Routing Problem with stochastic demands. We developed several genetic operators, included a local search and extended the algorithm with more advanced techniques. As this problem has a time-consuming solution evaluation function, which is based on dynamic programming, a multi-level evaluation scheme is applied to reduce the time spent for evaluations. Furthermore, a complete trie-based solution archive was implemented which stores all generated solutions in order to convert identified duplicate solutions into guaranteed new ones, which are also usually similar to the duplicate.

Extensive computational tests were performed, which show a significant improvement in the duration of calculating the fitness and consequently in the whole algorithm. These results demonstrate that the multi-level evaluation scheme is able to improve the results significantly in terms of running time and final solution quality. The experimental results after applying the solution archive indicate an improvement in some of the benchmark instances. In particular, using both the multi-level evaluation scheme and the solution archive lead to statistically significant better results in 2 instances compared to the configuration in which only the solution archive is used, which resulted in an improvement in 3 instances. Therefore it seems that in some of the test instances the time overhead of the operations in the solution archive does not pay off but that it can still be viable.

Compared to two algorithms from the literature which use the same set of instances as we do, revealed a strong indication that the developed genetic algorithm has the better performance. In all of the tested benchmark instances, the proposed algorithm finds solution with a higher quality in the same amount of computation time.

6.1 Future Work

There is a wide variety of possibilities and research directions to continue this work. As possible future work the following points can be considered.

In metaheuristics there are a plenty of well-known neighborhood structures described in the literature which can be used for solving a set of combinatorial optimization problems. In this thesis the 2-opt local search was considered. This can be extended by k-opt local search and be optimized for a better performance. Furthermore, any other search algorithms, which repeatedly trying to improve the current solution by looking for a better solution which is in the neighborhood of the current solution, can be used, e.g., an embedded tabu search approach.

The techniques developed in this thesis can also be used when additional constraints are considered. A maximal tour length, for example, can be respected by altering the solution evaluation. Then, a giant tour decoder has to be applied and the dynamic programming algorithm must be executed for each individual tour. Moreover, the algorithm, which has been developed in this work, can potentially be used in many real world problems involving stochastic routing problems, for example, the daily demand for cash at a bank's automatic teller machine. The maximal amount of cash that may be carried on a vehicle is dictated by security policy and corresponds to the vehicles' capacity.

List of Algorithms

3.1 A standard GA in pseudocode 12
3.2 A generic local search in pseudocode 14

4.1 Initialization 18
4.2 Optimized GA using 2-opt 19
4.3 Exact 2-opt 21
4.4 Objective function (fitness) of an a priori tour 22
4.5 Exact fitness evaluation using multi-level evaluation scheme 23
4.6 Inserting a solution candidate into the trie 27
4.7 Convert a duplicate solution 30
4.8 High-level description of the solution archive 30

List of Figures

4.1 2-opt move 20
4.2 2-opt considering exact evaluation parameters 20
4.3 An exemplary demand probability distribution and its different levels of approximation [BHR15]. 25

4.4 Inserting a solution into an empty trie data structure 28

List of Tables

5.1 Used parameter for capturing the result of computational tests on GA 33
5.2 Computational results on instances using default GA for population size of 100, 300, and 500 34
5.3 Objective mean and the objective geometric mean of different population size 34
5.4 Statistical results of increasing population size 35
5.5 Wilcoxon rank-sum results on population size 35
5.6 Comparison of applying optimization method on the population size of 500 . 36
5.7 Objective mean and the objective geometric mean of all runs after applying optimization method on the population size 500 37
5.8 Wilcoxon rank-sum test results on applying optimization method 37
5.9 Intermediate result of testing ML-ES 38
5.10 Intermediate result of testing ML-ES 38
5.11 Comparison of applying ML-ES with existing approaches 39
5.12 Objective mean and the objective geometric mean of applying ML-ES with existing approaches 40

Bibliography

[BBBB95] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[BBC$^+$06] Leonora Bianchi, Mauro Birattari, Marco Chiarandini, Max Manfrin, Monaldo Mastrolilli, Luis Paquete, Olivia Rossi-Doria, and Tommaso Schiavinotto. Hybrid metaheuristics for the vehicle routing problem with stochastic demands. *Journal of Mathematical Modelling and Algorithms*, 5(1):91–110, 2006.

[BDGG09] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing: an international journal*, 8(2):239–287, 2009.

[Bel54] Richard Bellman. The theory of dynamic programming. Technical report, DTIC Document, 1954.

[Ben82] J. L. Bentley. *Writing Efficient Programs.* Prentice Hall, 1982.

[BHR15] Benjamin Biesinger, Bin Hu, and Günther R. Raidl. A variable neighborhood search for the generalized vehicle routing problem with stochastic demands. In Gabriela Ochoa and Francisco Chicano, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2015*, volume 9026 of *LNCS*, pages 48–60. Springer, 2015.

[BK78] W Bialas and M Karwan. Multilevel linear programming. *State University of New York at Buffalo*, 1978.

[BR03] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[CL07] Christian H. Christiansen and Jens Lysgaard. A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. *Oper. Res. Lett.*, 35(6):773–781, 2007.

[CLM01] Jean-François Cordeau, Gilbert Laporte, and Anne Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research society*, pages 928–936, 2001.

[Cro58] Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

[DR59] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 10 1959.

[DT90] Moshe Dror and Pierre Trudeau. Split delivery routing. *Naval Research Logistics (NRL)*, 37(3):383–402, 1990.

[GH88] David E Goldberg and John H Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[GITN04] Peng Gang, Ichiro Iimura, Hidenobu Tsurusawa, and Shigeru Nakayama. A local search algorithm based on genetic recombination for traveling salesman problem. In Maarten Keijzer, editor, *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA, 26 July 2004.

[GLS96a] Michel Gendreau, Gilbert Laporte, and René Séguin. Stochastic vehicle routing. *European Journal of Operational Research*, 88(1):3 – 12, 1996.

[GLS96b] Michel Gendreau, Gilbert Laporte, and René Séguin. A tabu search heuristic for the vehicle routing problem with stochastic demands and customers. *Operations Research*, 44(3):469–477, 1996.

[GML08] Carlos García-Martínez and Manuel Lozano. Local search based on genetic algorithms. In Patrick Siarry and Zbigniew Michalewicz, editors, *Advances in Metaheuristics for Hard Optimization*, Natural Computing Series, pages 199–221. Springer, 2008.

[HHJL08] William Ho, George TS Ho, Ping Ji, and Henry CW Lau. A hybrid genetic algorithm for the multi-depot vehicle routing problem. *Engineering Applications of Artificial Intelligence*, 21(4):548–557, 2008.

[HS04] Holger H Hoos and Thomas Stützle. *Stochastic local search: Foundations & applications*. Elsevier, 2004.

[KK06] Vlasis K Koumousis and Christos P Katsaras. A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *Evolutionary Computation, IEEE Transactions on*, 10(1):19–28, 2006.

[Mit98] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[MMM14] Yannis Marinakis, Magdalene Marinaki, and Athanasios Migdalas. *A Hybrid Clonal Selection Algorithm for the Vehicle Routing Problem with Stochastic Demands*, pages 258–273. Springer International Publishing, 2014.

[Mos98] Gur Mosheiov. Vehicle routing with pick-up and delivery: tour-partitioning heuristics. *Computers & Industrial Engineering*, 34(3):669–684, 1998.

[MRV15] Jorge E Mendoza, Louis-Martin Rousseau, and Juan G Villegas. A hybrid metaheuristic for the vehicle routing problem with stochastic demand and duration constraints. *Journal of Heuristics*, pages 1–28, 2015.

[PGGM12] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2012.

[PS03] Russel J Petch and Said Salhi. A multi-phase constructive heuristic for the vehicle routing problem with multiple trips. *Discrete Applied Mathematics*, 133(1):69–92, 2003.

[RH10] G. R. Raidl and B. Hu. Enhancing genetic algorithms by a trie-based complete solution archive. In Peter Cowling and Peter Merz, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 6022 of *LNCS*, pages 239–251. Springer Berlin Heidelberg, 2010.

[SCP$^+$13] Nagesh Shukla, AK Choudhary, PKS Prakash, KJ Fernandes, and MK Tiwari. Algorithm portfolios for logistics optimization considering stochastic demands and mobility allowance. *International Journal of Production Economics*, 141(1):146–166, 2013.

[SCPB04] Martín Safe, Jessica Carballido, Ignacio Ponzoni, and Nélida Brignole. On stopping criteria for genetic algorithms. In *Advances in Artificial Intelligence–SBIA 2004*, pages 405–413. Springer, 2004.

[SGV11] Geetha Shanmugam, Poonthalir Ganesan, and PT Vanathi. Meta heuristic algorithms for vehicle routing problem with stochastic demands. *Journal of Computer Science*, 7(4):533, 2011.

[SP00] Dimitrios Sariklis and Susan Powell. A heuristic method for the open vehicle routing problem. *Journal of the Operational Research Society*, 51(5):564–573, 2000.

[Spe13] Simon Sperl. A Genetic Algorithm for the Stochastic Vehicle Routing Problem, 2013. Bachelor thesis, Vienna University of Technology.

[YMB00] Wen-Huei Yang, Kamlesh Mathur, and Ronald H Ballou. Stochastic vehicle routing problem with restocking. *Transportation Science*, 34(1):99–112, 2000.