Isfandyar Khan Mian

Combining Vehicle Routing Optimization and Container Loading Optimization

Master's Thesis in Information Technology

July 9, 2020

University of Jyväskylä

Department of Mathematical Information Technology

Author: Isfandyar Khan Mian

Contact information: `isfando@yahoo.com`

Supervisors: Prof. Olli Bräysy, and Asst. Prof. Michael Cochez

Title: Combining Vehicle Routing Optimization and Container Loading Optimization

Työn nimi: Ajoneuvojen reitityksen optimoinnin ja konttien lastausoptimoinnin yhdistäminen

Project: Master's Thesis

Study line: Web Intelligence and Service Engineering

Page count: 93+0

Abstract: Vehicle routing optimization and container loading combined would produce millions of queries for the remaining capacity of the vehicles. In this situation, these approximate methods for finding the remaining capacity of the vehicle's container are investigated. These methods reduce the time needed to approximate the remaining capacity in vehicles and will hence accelerate the overall optimization process. In this thesis we consider a solution to improve the accuracy of real-world vehicle routing optimization problems. Simple capacitated vehicle routing optimization does not capture any information about the packing of objects except by deducting the volume of the packed objects from the container's volume. Bin packing during the routing optimization is usually slow. We combine a very fast approximation algorithm for 3D bin packing with vehicle routing optimization to speed up the whole process. Vehicle routing combined with the 3D container loading problem creates new kinds of challenges. The problem was introduced in Gendreau et al. 2006 where 3D loading space replaces the scalar capacity of the vehicles. The container loading problem attempts to obtain the best possible utilization of space, while the vehicle routing problem is concerned with finding the minimum-cost or minimum-distance route in transportation. The combined problem is about loading boxes with different symmetry into rectangular containers of the vehicles used in delivery. This problem is extremely hard because it is a combination of the two problems mentioned above, which are both NP-hard [Gendreau et al. 2006, Pisinger

2002]. Finding an exact solution for this problem is infeasible since even solving a small instance of bin packing problem alone would require more computing resources as feasible (Martello, Pisinger, and Vigo 2000). To handle this situation approximation algorithms are used as it is often not necessary to find the optimal solution for the bin packing problem. An approximate solution that is close to optimal and computed with the help of reasonable resources and time is considered a good solution. When vehicle routing optimization and container loading are combined, a high number of queries for the remaining capacity of the vehicles are performed. In this thesis we exploit this fact and perform experiments with approximate methods for finding the remaining capacity of the vehicle's container in a fast but approximate way. In our experiments we use a slight modification of the 3D bin packing algorithm called Largest Area First Fit (LAFF) (Gürbüz et al. 2009) as a rough but fast means to determine the remaining capacity in the containers during the vehicle routing optimization process. A bounding box is used for objects which are not rectangular in shape, such as cylindrical shapes. The LAFF algorithm carries the placement of the boxes such that those with the largest surface area are placed first while keeping the height minimum from the floor of the container. The box which covers the largest ground area of the container is placed first followed by subsequent boxes that are stacked in the remaining space at the same level, the boxes with the greatest volume first. Then the level is increased and the process repeated. Boxes are rotated such that they have the largest possible footprint. This algorithm works exceptionally fast when the number and variety of the objects to be packed are small. During the LAFF stage, all real-world bin packing constraints e.g. the weight of the boxes, loading priorities, orientation, stacking, the distribution of weight in different parts of the container, stability, etc. are ignored to gain as much speed as possible.

Keywords: Vehicle Routing Optimization, Vehicle Loading Optimization, Container Loading Optimization, Logistics Optimization

Suomenkielinen tiivistelmä: Ajoneuvojen reitityksen optimointi ja konttien lastauksen optimointi yhdessä tuottaisivat miljoonia kyselyjä ajoneuvojen jäljellä olevasta kapasiteetista. Tässä tilanteessa tutkitaan likimääräisiä menetelmiä ajoneuvon kontin jäljellä olevan kapasiteetin löytämiseksi. Nämä menetelmät vähentävät aikaa, joka tarvitaan ajoneuvojen jäljellä olevan kapasiteetin arviointiin, ja nopeuttavat siten yleistä optimointiprosessia. Tässä opinnäytetyössä käsittelemme ratkaisua reaalimaailman ajoneuvojen reitityksen optimointiongelmien tarkkuuden parantamiseksi. Yksinkertainen kapasitiivisen ajoneuvoreitityksen optimointi ei käytä mitään muuta tietoa esineiden muodosta kuin vähentää pakattujen esineiden tilavuuden kontin tilavuudesta. Konttien lastaus reitityksen optimoinnin aikana on yleensä hidasta. Prosessin nopeuttamiseksi, yhdistämme ajoneuvojen reitityksen optimointiin erittäin nopean 3D-Konttien lastaukseen likimääräisen algoritmin. Ajoneuvojen reitityksen ja 3D-konttien lastausongelman yhdistäminen luo uudenlaisia haasteita. Näiden yhdistelmä esiteltiin artikkelissa Gendreau et al. 2006, jossa ajoneuvojen skalaarikapasiteetti korvattiin kolmiulotteisen suorakaiteen muotoisella lastaustilalla. Konttien lastausongelmalla yritetään saada aikaan paras mahdollinen tilankäyttö, kun taas ajoneuvojen reititysongelman tarkoitus on löytää pienimmän kustannuksen tai pienimmän kuljetun etäisyyden reitti. Yhdistetyssä ongelmassa on kyse eri tavoin symmetristen laatikoiden lastaamisesta toimituksessa käytettyjen ajoneuvojen, suorakaiteen muotoisiin kontteihin. Tämä ongelma on erittäin vaikea, koska se on yhdistelmä kahdesta edellä mainitusta ongelmasta, jotka molemmat ovat NP-vaikeita [Gendreau et al. 2006, Pisinger 2002]. Eksaktin ratkaisun löytäminen tälle ongelmalle ei ole käytännöllistä, koska edes pelkän pienen konttien lastausongelman ratkaiseminen edellyttäisi enemmän laskentaresursseja kuin mahdollista (Martello, Pisinger, and Vigo 2000). Tämän tilanteen käsittelemiseksi käytetään likimääräisiä algoritmeja, koska usein ei tarvitse löytää optimaalista ratkaisua konttien lastausongelmaan. Hyväksi ratkaisuksi luetaan likimääräinen ratkaisu, joka on lähellä optimaalista ja jonka laskemiseen on käytetty kohtuullinen määrä resursseja ja aikaa. Kun ajoneuvojen reitityksen ja konttien lastauksen optimointi yhdistetään, tarvitaan suuri määrä kyselyjä ajoneuvojen jäljellä olevasta kapasiteetista. Tässä opinnäytetyössä hyödynnetään tätä tosiasiaa ja suoritetaan kokeita likimääräisillä menetelmillä ajoneuvon kontin jäljellä olevan kapasiteetin löytämiseksi nopeasti, mutta likimääräisesti. Kokeissamme käytämme pientä modifiointia 3D-konttien latausalgoritmiin. Tämän algoritmin nimi on Suurin Alue Ensimmäisenä (LAFF) (Gürbüz et al. 2009). Tämä algoritmi on karkea mutta nopea keino konttien jäljellä oleva kapasiteetin määrittämiseen ajoneuvon reitityksen optimointiprosessin aikana. Esineiden jotka eivät ole suorakaiteen muotoisia, vaan esimerkiksi lieriömäisiä, arviointiin käytetään laatikkoa. LAFF-algoritmi sijoittaa ensin laatikot, joilla on suurin pinta-ala, ja minimoi korkeuden säiliön pohjasta. Laatikko joka peittää suurimman maa-alueen asetetaan ensin, ja tämän jälkeen seuraavat laatikot pinotaan jäljellä olevaan tilaan samalla tasolla, edeten aina laatikolla jolla on suurin tilavuus. Sitten tasoa nostetaan ja prosessi toistetaan. Laatikot käännetään siten, että niillä on suurin mahdollinen jalanjälki. Tämä algoritmi toimii poikkeuksellisen nopeasti, kun pakattavien kohteiden lukumäärä ja monimuotoisuus on pieni. LAFF-vaiheen aikana kaikki reaalimaailman pakkausrajoitukset, kuten laatikoiden paino, painon jakautuminen säiliössä, lastausprioriteetit, suuntaus, pinoaminen, vakaus jne. jätetään huomioimatta, mahdollisimman nopean pakkaamisen saavuttamiseksi.

Avainsanat: Ajoneuvojen reitityksen optimointi, ajoneuvojen lastauksen optimointi, konttien lastauksen optimointi, logistiikan optimointi

Preface

I would like to thank my family for their continuous support during the process of my studies. Their love and faith in my abilities have been instrumental in reaching this point in my life. I am also grateful to my daughter, Zara Khan, for being a positive impact during this process through her joyous and motivating energy. I appreciate my sister, Hareem Hilal's support in reviewing the final draft.

I acknowledge Dr. Olli Bräysy for believing in me and giving me valuable learning opportunities. I would like to extend my sincerest gratitude to Dr. Michael Cochez; an inspiring mentor and a kind friend. His advice and guidance got me through tough times and deadlocks. In the end, I would like to thank my colleague Pekka Hotakka for giving his opinions on certain topics in this thesis.

Jyväskylä, July 9, 2020

Isfandyar Khan Mian

List of Figures

List of Tables

Contents

1 Introduction

This introductory chapter provides the motivation, scope, research goals, research questions and summary of the contribution of the thesis.

1.1 Motivation

Vehicle routing optimization is an important but computationally complex task. Nowadays, it is difficult to handle large amounts of customers in the delivery and logistics business in a cost-effective way without vehicle routing optimization. A major problem in vehicle routing optimization is the loading of vehicles used in the delivery of the goods. It is not possible to use the full capacity of vehicles because of different loading constraints for different packages of goods. We can approximate the remaining capacity of container heuristically within a reasonable time. However, the overall process remains time-consuming. Vehicle routing and container loading (packing) are highly interdependent. For example, if the loading space of a truck has been efficiently utilized but the routing and transport process is weak, there will be no value-added or vice versa. That is why solving both problems at once has huge practical applications in logistics and transportation, especially in those cases where the shippers need to deal with large or fragile items like home appliances and furniture like sofas, etc. (Iori, Salazar-González, and Vigo 2007).

Some of the benefits of integrating the routing and loading are given below:

- Quality of packaging and transport process increases greatly.
- On-time delivery of goods to the customer.
- Undamaged and in-order arrival of the goods.
- Better loading of high-cost, high-risk and different-shaped goods like furniture.

1.2 Scope

The combination of Vehicle Routing Problem and Container Loading creates a new kind of problem. This problem is about loading boxes with different symmetry that are to be packed into rectangular containers of the vehicles used in delivery. This problem is extremely hard because it is a combination of two problems, vehicle routing problem and packing problem, specifically a 3D-Bin Packing Problem (3D-BPP), which are both NP-hard problems. This problem can be solved by finding a loading plan that can contain all the boxes which need to be delivered to specific customers in a given route. It should be kept in mind that there are constraints such as the total weight of goods must not exceed the vehicles total weight capacity. If there are fragile items, then their placement must be such that non-fragile items are not placed over them. The items might need to be supported totally or partially while being placed in the container. The containers need to be loaded in a way that facilitates unloading without shifting other customer's demands.

The general container loading is too hard to solve. Hence, only solutions to a constrained version of the problem are presented. The constraints, in this case, means that only rectangular, square or cylindrical shaped objects can be loaded into the container. The capacity checker would be fast which will introduce a certain level of error in the process due to approximation of the current space in the container. Weight limitations of the container cannot cross the threshold even if the cumulative volume of given objects can be placed inside the container.

1.3 Research Goals of the Thesis

We are trying to find a solution to a combination of vehicle routing and packing problem. The problem we are trying to solve is extremely hard because it is a combination of two types of NP-hard problems. That is why an exact solution approach for realistic scenarios is not quite efficient for this problem yet. In order to solve three-dimensional Capacitated Vehicle Routing, the so-called Single Vehicle Loading Problem, needs to be solved in numerous times. In Knapsack Problems, chapter 1, by Kellerer, Pferschy, and Pisinger 2004a, it is discussed that the attempts to compute optimal solutions for the knapsack problem are not entirely satisfactory, especially if there is a very limited time and space available. Knapsack can be considered a particular case of bin packing where the weight and value of items are the dimensions taken into account. In the context of algorithmic performance, time and space are running time on a computer, and space is the amount of computer memory to solve a given problem respectively. It is clear that all algorithms do not perform equivalently while computing an optimal solution. It would seem intuitive that large computational time can be reduced by an algorithm computing approximate solutions even though they may not necessarily be optimal. It should also be noted that the difficulty level of the algorithm also plays an important role. An easy algorithm is less costly to implement as compared to a complex algorithm. In practice this is not very easy to measure that which algorithm is easy and which is difficult because the implementation cost is also dependent on programming experience and computational environment.

So far, the best-obtained solutions for three-dimensional Capacitated Vehicle Routing Problem are based on heuristic solutions. There are some meta-heuristics approaches also available which are mainly based on Tabu-search. There are other solutions based on the Ant colony, Bee mating and hybrid solutions available in the literature. However, the overall process remains time-consuming. The reason is that in large instances of vehicle routing optimization problems there would be millions of queries for finding the remaining capacity of the vehicles. In this thesis approximate methods for finding the remaining capacity of the vehicle's container are investigated. These methods reduce the time needed to approximate the remaining capacity in vehicles and will hence accelerate the overall optimization process. Another advantage is the presence of human checking after the optimization results are obtained. There are practices in place in the industry that use manual human checking on top of the optimization process. In the case of a small error in the bin packing optimization result, things can still be handled by making small changes to the packing process or delivery process. One example of this is the usage of temporary rented vehicles to deliver goods on all the routes. These temporary vehicles are not part of the fleet when the optimization is being run.

1.4 Research Questions

The following are the research questions that this thesis tries to answer:

1. How should Heterogeneous Fleet of Pick Up and Delivery Problem with Time Windows and three-dimensional Loading Constraints (3L-HFCVRPTW) be combined with

a practical and fast 3D packing solution?

2. How should the combination of 3L-HFCVRPTW and 3D bin packing be executed in a practical computational time limit?

1.5 Summary of Contribution

We have integrated a 3D bin packing algorithm which is a slight variant of Largest Area Fit First (LAFF) minimizing height with a variant of 3LCVRP, which is a Heterogeneous Fleet of Pick Up and Delivery Problem with Time Windows and Three-Dimensional loading constraints (3L-HFCVRPTW). The inclusion of 3D packing process has given us an estimate close to the real world packing while keeping the computational time limits practical.

2 Optimization

2.1 Introduction

The process of optimization is used to achieve the best possible results in a given situation as observed in Optimization An Introduction, chapter 1, by Astolfi 2006 and Fundamentals of Optimization, chapter 1, by Rockafellar 2006, which are the basis of discussion in this section. In different fields such as design, construction and maintenance crucial decisions need to be made on day to day basis. The goal is to achieve maximum benefit by putting minimum effort. Optimization is generally the maximization and minimization of a function.

2.1.1 Description of an Optimization Problem

The description of problem relating to optimization or mathematical programming can be given as follows.

Finding

$$\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n \tag{2.1}$$

minimizing

$$f(x) \tag{2.2}$$

complying with the following constraints.

$$g_j(x) \leq 0, \quad \text{for } j = 1, \ldots, m. \tag{2.3}$$

and

$$l_j(x) = 0, \quad \text{for } j = 1, \ldots, p. \tag{2.4}$$

Design vector is exhibited by variable **x**, a given system contains a number of quantities which may be employed as a variable in the designing of the optimization model. Among such quantities those that can be considered variables are referred to as design/decision variables, and forms part of vector **x**. The objective function is denoted by $f(x)$, the inequality constraints are expressed by $g_j(x)$ while the equality constraints are expressed by $l_j(x)$. The constraints $p+m$ and variables n are not necessarily relevant. The problem will be that of an unconstrained optimization if $p+m$ results in zero. There are other types of design constraints related to vehicle routing and bin packing which will be discussed later on in the thesis.

2.1.2 Design Constraints

The design variables cannot be selected arbitrarily in the construction of the optimization model. They need to satisfy certain restrictions called design constraints. A feasible point is any point that fulfills all the constraints in an optimization problem. An optimal point is one that optimizes the value function given the constraints. Design constraints might be performance limitation or behavior of the system or physical limitations of the system. Some of the types of design constraints are given below.

Equality and inequality constraints The limitations of the form $f_i(x) = c_i$, $f_i(x) \leq c_i$ or $f_i(x) \geq c_i$ for certain functions f_i on $\mathbb{R}^n$ and constants c_i in $\mathbb{R}$. For example a budget constraint of 20 dollar while shopping in a store will allow us to buy 5 kg of oranges and 2 kg of apples if the price of one kg oranges is 2 dollars and the price of one kg apples is 5 dollars.

Range constraints The limitations which restrict the values of some decision variables to remain within specific closed intervals of $\mathbb{R}$. An important example of this is the non negativity constraint in which some variable x_j may be allowed to only take values ≥ 0; with $[0,\infty)$ as interval. For example, the speed of the car should be between the range of 80 km/h to 100 km/h.

Linear constraints The limitations which cover range constraints and condition of the form $f_i(x) = c_i$, $f_i(x) \leq c_i$ or $f_i(x) \geq c_i$, in which the function is linear. Here linear means that the function can be expressed as the sum of constant-coefficient times the variables $x_1, \ldots, x_n$.

Data parameters In addition to decision variables problem statements use symbols designating "given" constants and coefficients. They represent some other perspective of the optimization problem aside from constraints.

2.1.3 Objective Function

The aim of the classical design procedure is to find a design satisfying all of the constraints i.e. a design that is acceptable.

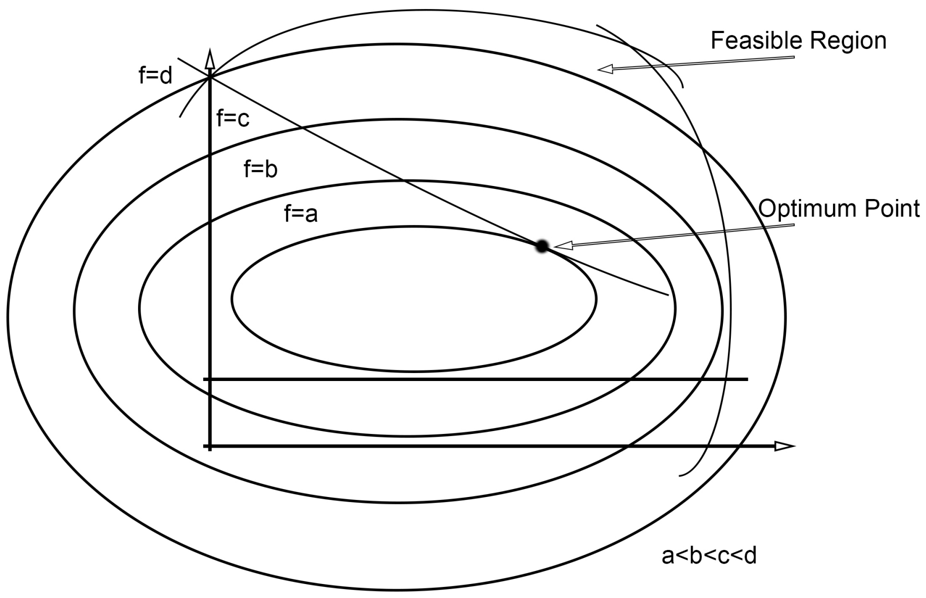

Figure 1: Optimization Illustration

Generally there can be several designs that satisfy the constraints, and the purpose of constructing the optimization model is to find the best possible design. We need criteria in order to carry out a comparison between various designs. This criteria amount to an objective function when it is illustrated in terms of design variables. Physical or economical considerations generally have an impact on the objective function. One of the most crucial decisions in the entire design process is the selection of the objective function, as usually there is a trade-off between performance and cost or between performance and reliability. In multi-objective optimization problem, the sum of various objective functions resulting in a cost function may be used in reaching an approximate solution of different criteria.

There are numerous ways in which an optimization problem may be classified:

- Constrained vs unconstrained.
- Considering the kind of objective function and the constraints an optimization problems may be categorized into linear, quadratic, polynomial and non-linear.
- Optimization problems can be categorized into integer or real-valued, and deterministic or stochastic based on design variables' values.

2.2 Asymptotic Analysis

Asymptotic analysis of an algorithm, refers to defining the mathematical depiction of its run-time performance. The approach is used to calculate the best, average and worst case scenario of an algorithm (Knuth 1976). Some of the asymptotic notations are given as follows:

Worst-case performance Worst-case performance is generally represented by $O(n)$ (called Big Oh notation) and measures the longest amount of time required for an algorithm to execute.

$O(f(n)) = g(n): \; there\,exists \;\; c \geq 0 \;\; and \;\; n_\theta \;\; such\,that \;\; g(n) \leq c.f(n) \;\; for \;\; all \;\; n > n_\theta.$

Best-case performance Best-case performance is generally represented by $\Omega(n)$ (called Omega notation) and measures the best case time required for an algorithm to execute.

$\Omega(f(n)) \geq \{g(n): there\,exists \;\; c > 0 \;\; and \;\; n_\theta \;\; such\,that \;\; g(n) \leq c.f(n) \; for\,all\,n > n_\theta.\}$

Average case performance Average case performance is generally represented by $\theta(n)$ (called Theta notation) and measure the lower and upper bound of time taken by an algorithm to execute.

$\theta(f(n)) = \{g(n) \, if\,and\,only\,if \;\; g(n) = O(f(n)) \;\; and \;\; g(n) = \Omega(f(n)) \;\; for\,all \;\; n > n_\theta.\}$

2.2.1 Computation Complexity

The following discussion is mostly based on observations from Darapuneni 2012 and Computers and Intractability, chapters 1,2 and 5, by Garey and Johnson 1979.

P The class of decision problems which can be solved by an algorithm in polynomial time. This means that the running time of the algorithm is bounded by a polynomial of input size. Let T(n) be the execution time of the algorithm for input size n. Then there exists a constant k such that the execution time $T(n)$ is $O(n^k)$.

NP The class of decision problems, for which the ”yes" answers have proofs verifiable in polynomial time by a deterministic Turing machine. NP stands for “nondeterministic polynomial time”. This means that there is an efficient algorithm that gives a solution to the problem instance, can we verify the correctness of the solution. The class P is contained in class NP i.e. $P \subseteq NP$. It is an open problem whether $P = NP$. This is called the P vs NP problem. This means that whether or not NP problems have deterministic polynomial time solutions.

NP-Hard The class of problems is said to be in NP-hard if they are at least as hard as the hardest problem in NP. It is not necessary for an NP-hard problem to be a decision problem and to be in NP. A problem X is considered NP-hard, if there is an NP-complete problem Y, such that Y is reducible to X in polynomial time.

NP-Complete The class of problems is said to be NP-complete where a decision problem belongs to NP and it is also NP-hard. In other words it is the set of all problems X in NP for which it is possible to reduce any other NP problem Y to X in polynomial time. The importance of solving an NP-complete problem is that if we are able to find an algorithm to solve an NP-complete problem in polynomial time then we can solve every other NP problem in polynomial time. They are the hardest problems in NP.

2.3 Exact Methods

Exact algorithms are used to find the optimal value for an optimization problem. This kind of algorithm can’t run in worst-case polynomial time, unless P = NP, but researchers have been trying to find exact algorithms of exponential running time with a low base. The discussion in this section is based on Vehicle Routing, chapter 2, by Semet, Toth, and Vigo 2014 and El-Sherbeny 2010. As an example, we would look into exact methods to solve the Vehicle Routing Problem with Time Windows (VRPTW). The purpose of solving vehicle routing problems is the minimization of a cost function by using a given set of vehicles to find a way for visiting a given set of locations. In such instances the cost function is usually the total driving distance or total driving time. Vehicle routing optimization comes in the category of problems called NP-hard. Solving NP-hard optimization problem has been quite a challenging task for researchers even before the discovery of the concept of NP-hardness (Ropke 2005). Researchers have made significant progress recently in this field but for most of the problem only fairly small instances can be solved to optimality consistently. Generally exact methods for solving the VRPTW are categorized into three classes: Lagrange relaxation, column generation and dynamic programming. Also variants of integer programming can be used to solve optimization problems. Exact methods are usually very inefficient, time-consuming and unable to find optimal solutions for large problem instances (El-Sherbeny 2010).

2.3.1 Lagrange Relaxation

Different research papers have used different approaches to apply Lagrange relaxation-based methods e.g. Lagrange relaxation applied subsequently to variable splitting, Lagrange relaxation applied subsequently to K-tree approach etc. If for a graph there are $n+1$ nodes then the set $n+K$ edges will be K-tree that is spanning the graph. In this context spanning the graph means traversing through all nodes of the graph with the minimum possible number of edges. In Kohl and O. Madsen 1997, objective function provides for the minimization of costs.

$$\min \sum_{k\in V}\sum_{i\in N}\sum_{j\in N} c_{ij}x_{ijk},$$

subject to following constraints,

$$\sum_{k\in V}\sum_{j\in N} x_{ijk} = 1, \forall i \in C, \tag{2.1}$$

$$\sum_{i\in C} d_i \sum_{j\in N} x_{ijk} \leq q, \forall k \in V, \tag{2.2}$$

$$\sum_{j\in N} x_{0jk} = 1, \forall k \in V, \tag{2.3}$$

$$\sum_{i\in N} x_{ihk} - \sum_{j\in N} x_{hjk} = 0, \forall h \in C, \forall k \in V, \tag{2.4}$$

$$\sum_{i\in N} x_{i,n+1,k} = 1, \forall k \in V, \tag{2.5}$$

$$\begin{aligned} &s_{ik} + t_{ij} - K(1 - x_{ijk}) \\ &\leq s_{jk}, \forall i \in N, \forall j \in N, \forall k \in V, \end{aligned} \tag{2.6}$$

$$a_i \leq s_{ik} \leq b_i, \forall i \in N, \forall k \in V, \tag{2.7}$$

$$x_{ijk} \in \{0,1\}, \forall i, \in N, \forall k \in V. \tag{2.8}$$

If constraint 2.1 is Lagrangian relaxed i.e. relaxing the constraint ensuring that each customer is served exactly once and the objective function can be written as

$$min \sum_{k\in V}\sum_{i\in N}\sum_{j\in N} (c_{ij} - \lambda_j)x_{ijk} + \sum_{j\in C} \lambda_j$$

subject to 2.2 to 2.8. Here the Lagrange multiplier λ_j subject to the constraint makes sure the provision of service to customer j.

In Fisher, Jörnsten, and Madsen 1997, two optimization methods are given for solving the vehicle routing problem in an optimal fashion. The problem is expressed as a K-tree with degree 2K on the depot. The two methods are a K-tree relaxation having time windows as a side constraint and a Lagrangian decomposition consisting of variable splitting that divides the problem into two subproblems. One will be a semi-assignment problem and the other a series of shortest path problems having the constraints of time windows and capacity.

2.3.2 Column Generation

Column generation is a useful algorithm for providing efficient solutions for larger linear problems. The problem under consideration is divided into two problems namely the master problem and the subproblem. In the event that there are excessively numerous variables in a linear program that are to be tackled explicitly, such a linear program can be regarded as a master problem containing a smaller number of variables and calculating its solution. Next, we can add one or more unexplored variables as a subproblem, that may enhance the solution

In practical implementation, a master problem is solved with the aim to use the solution to get dual prices for each of the constraints. The objective function of the subproblem uses this information and then it is solved. If the subproblem gives negative objective value, it signals the identification of a variable that have a negative reduced cost. This prompts the addition of a variable to the master problem, which is resolved. The process of resolving the master problem repeats the initial cycle until we obtain a reduced cost with non-negative value. Thus we arrive at the conclusion that the master problem has been optimally solved (Knapsack Problem, chapter 15, by Kellerer, Pferschy, and Pisinger 2004b).

Agarwal, Mathur, and Salkin 1989 addressed the solution of VRP through column generation, while Desrosiers, Soumis, and Desrochers 1984 used the column generation approach to solve the multiple traveling salesperson (m-TSP) with time windows.

2.3.3 Dynamic Programming

Dynamic programming is a technique to solve complex optimization problems. The problem is broken down into many small problems. Each small problem is solved once and the solution is stored in a memory-based data structure. Next time if we again face the same subproblem that was solved earlier, we use the result from memory instead of recomputing the whole solution to the big problem from scratch. This technique of storing solutions to subproblems is also called "memoization". Memoization is similar to recursion with a modification that will check the lookup table for a precomputed solution before computing a solution.

Kolen, Kan, and Trienekens 1987, addressed VRPTW through the approach of dynamic programming. The algorithm of Kohl and O. B. G. Madsen 1997 use branch-and-bound to achieve optimality. A branch and bound algorithm consists of different candidate solutions in state-space search. The algorithm traverses through branches of the tree, which represents a subset of the solution. The branch is checked with upper and lower estimated bounds on the optimal solution before enumerating its candidate solutions. The branch is discarded if it can not give a solution that is better than the current best selection. In Rothlauf 2011, the process of adding additional constraints to the original problems is called branching. This can be modeled using hierarchical tree structures. The process of removing generated subproblems from further consideration is called bounding. Once a subproblem is removed from the tree in the bounding process, it will not be considered anymore and is not decomposed into further subproblems.

2.3.4 Integer Programming

In integer programming, a mathematical optimization problem restricts all the variables to be integers (Integer and combinatorial optimization, chapter 1, by Wolsey and Nemhauser 1999). In most of the cases, it can refer to Integer Linear Programs (ILP). MILP is a variant of an integer programming problem and stands for Mixed-Integer Linear Programming (MILP). MILP involves problems in which only some of the variables are constrained to be integers, while others are allowed to be non-integers. In general there are discrete optimization problems, and many of them can be formulated and solved using MILP solvers such as Gurobi. Gurobi is a commercial optimization solver for linear programming, quadratic programming and mixed integer linear programming, etc. Other alternatives to Gurobi are CPLEX, Mathematica and LINDO etc.

In Bula et al. 2016, a two-stage approach is formulated to transport hazardous material using Heterogeneous Vehicle Routing Problem (HVRP). Estimation and analysis of risks is an important aspect in the transport of hazardous material. One of the main focuses is the selection of the safest routes. In the first stage, the objective function uses a linear approximation of the total routing risk. The routing risk measure is assumed as a non-linear function of the truckload, which is approximated by means of two different piecewise linear functions (PLF). In the second stage to solve the overall risk optimization problem, MILP is used to integrate the best piecewise linear approximations of the routing risk. The MILP model can optimize risk for instances with 20 nodes in a practical time limit. Above 20 the CPU running time can exceed 15 hours e.g. in cases with 50 nodes.

Dondo and Cerdá 2007 developed three-stage heuristics for multi-depot routing problems involving time windows and vehicles. When applied to VRPTW, their clustering algorithm is capable of providing a solution for instances with a maximum of 25 nodes by restructuring MILP. In Çetinkaya, Karaoglan, and Gökçen 2013, a combination of Mixed Linear Programming (MIP) and Memetic Algorithm (MA) is used for solving two-stage vehicle routing problems with time windows. In Simbolon and Mawengkang 2014, a Mixed-Integer Approach (MIP) is presented for a variant of vehicle routing problem with time windows. In this variant some routes can be avoided. This is designed by checking edge traversing frequency. If the frequency is high then the option of sub-route is not opted so as to eliminate heavy traffic. The experimental results show that MIP formulation works well for instances having up to 50 nodes. In Aggarwal and Kumar 2019, a Mixed Integer Programming (MIP) is utilized to solve the Vehicle Routing Problem with Time Windows (VRPTW). The time window is considered a hard constraint. A new mathematical model of MIP is formulated and implemented in CPLEX. CPLEX is mathematical optimization solver by IBM which is used to solve integer programming problems and very large linear programming problems amongst a variety of other problems (Mittelmann 2007).

2.4 Approximation Algorithm

Approximation algorithms are used to find approximate solutions to optimization problems and they are mostly used to solve NP-hard problems. Thus unless P = NP, there exists no efficient algorithms to find optimal solutions to NP-hard problems. This discussion is inspired by Randomized Algorithms, chapter 1 and 2, by Motwani and Raghavan 2010 and Knapsack Problem, chapter 6, by Kellerer, Pferschy, and Pisinger 2004c.

In principle, any algorithm that produces a good enough solution is an approximate algorithm. A feasible approximate algorithm would produce an approximate solution that is closer to the optimal solution. Additionally, it is desired to inquire about the size of the deviation between the approximate solution and the optimal solution.

Let us take the example of bin packing in which the exact solution takes exponential time. This increases the difficulty in finding an exact solution. It would need more resources and time even for a small instance of bin packing problem. In order to handle this situation in a better way, approximation algorithms are used. In many instances, it is not necessary to find the optimal solution for the bin packing problem. An approximate solution that is close to the optimal solution and computed with the help of reasonable resources and time is considered a good solution.

Generally, an approximation algorithm may sometimes produce almost optimal solutions on specific data sets, but on other occasions it will produce solutions of inferior level. In such a situation, the worst-case performance of an approximation algorithm is studied. Another approach to measure the distance is an absolution performance guarantee. In all instances of the problem, it can be obtained by the maximum difference between the values of the optimal and approximate solution. Suppose an approximation algorithm A calculates the solution value of a problem instance I as $z^A(I)$, while the value of the optimal solution for I is given by $z^*(I)$. When we are dealing with the approximation of problems where the goal is to maximize the objective (knapsack) we have $z^A(I) \leq z^*(I)$.

Definition: A represents an approximation algorithm of absolute performance guarantee k, $k > 0$, if

$$z^*(I) - z^A(I) \leq k$$

is true for all problem instances I.

A relative performance guarantee is one of the ways to calculate the distance between approximate and optimal solution as it puts a bound on the maximum ratio between them. This approach is more plausible than the absolution performance guarantee as it does not depend on the magnitude of the value of the objective function.

Furthermore, for a given optimization problem P, there exists an algorithm A such that for any instance I, it computes solution $z^A(I)$, we say that this A provides an r-approximation solution for problem when for any instance I

$$\frac{1}{r} \leq \frac{z^*(I)}{z^A(I)} \leq r$$

Where $z^*(I)$ is the optimal solution for instance I of problem P. Here A is said to be a r-approximate algorithm.

2.5 Heuristic Methods

Heuristics is any approach to find the solution of a problem using a practical method not guaranteed to be optimal or perfect, but good enough for the immediate goals. Heuristics can be any clever approach to solving a problem that gives us a close enough solution to the optimal solution. They play an important role in finding a satisfactory solution to NP-hard problems. They use an educated guess or an intuitive judgment to solve a problem. The problem is divided into small parts and then tried to be solved in an iterative fashion. The advantage of using heuristics to solve a problem is that they can be quite flexible and can easily be adapted for different variants of a problem. Heuristics might work quickly and efficiently but their quality can not always be measured accurately. The quality of the heuristic function plays an important role in the performance of heuristic. Some of the techniques for improving quality of heuristic function is relaxing constraints of the problem, using a pattern database to storing precomputed solution costs for subproblems, or learning from the experience of the problem domain (Artificial Intelligence, chapter 3, by Russell and Norvig 2003a). The performance quality of heuristics can be assessed by comparing it with the results of benchmark tests for other heuristics used for solving the same problem. This kind of performance testing can't give absolute results because the benchmark test represents a small set of the input that our heuristic can face in a deployed scenario.

A heuristic might perform well in a similar class of problems but it might perform poorly for another class of problems. "No Free Lunch" theorem proposed in Wolpert and Macready 1997, states that there is not one master model that works best for every problem. So we can have a heuristic that performs better for a generic case, but it will perform poorly for specific cases. Likewise, a heuristic for a specific case will not be as good as some heuristic for generic cases.

The speed of heuristic-based algorithms is important in real-world applications. In some scenarios, the time for constructing or reconstructing a part of the solution is critical. It can happen if the circumstances change while following a certain plan or if the requirement of a customer for the transportation task changes.

2.5.1 Neighbourhoods

The neighborhood of a solution S consists of a set of solutions, N(S), that can be produced with a single change to S. In general, a neighborhood is a (potentially infinite) set of points 'close' to our current solution. The neighborhoods of the current solution are explored and a move to a new solution is made if and only if an improvement is made. In the context of vehicle routing, neighborhoods problem are classified into two main classes: Single-Route Improvements and Multi-Route Improvements as mentioned in Laporte and Semet 2001. They can also be called Single-Route neighborhoods and Multi-Route neighborhoods respectively. In single-route neighborhoods only one route is under consideration at a time for moves while multi-route neighborhoods do the same between two or more routes. Thus multi-route neighborhoods are capable of making substantial modifications to a solution.

2.5.2 Local Search Heuristics

Local search algorithm tries to solve a problem by moving from solution to solution in the space of candidate solutions. It applies local changes to the current solution until a solution deemed optimal is found or a processing time-bound is reached. The solution found to be optimal may not be globally optimum. In local search heuristic we have always got a current solution. The current solution is modified if the evaluation signals an improvement in the solution. The term improvement heuristic as discussed in Laporte and Semet 2001, refers to a local search heuristic that only makes such moves that will result in the improvement of the value of the solution's objective function. In some variants of local search heuristic, the current solution can be modified even if evaluation signals a negative result. This is done in the hope of finding a much better solution after a few steps in the solution search space.

2.5.3 Metaheuristics

Metaheuristic is a higher level heuristic used to select another heuristic that may provide a better solution to an optimization problem. It is mainly a subfield of stochastic optimization and combines a factor of randomness with heuristics. A metaheuristic is a master plan for guiding and modifying subordinate heuristics by an intelligent combination of various concepts for exploring the search space. It is useful especially when we have very little or incomplete information in advance or limited computational power to know the global optimum solution to a problem (Essentials of Metaheuristics, chapter 1, by Luke 2013). Metaheuristic is about optimizing the solution of a problem towards a better state but it does not ensure finding the globally optimum solution. The advantage of metaheuristics is the use of less computational power as compared to other optimization solution methods.

2.5.4 Construction Heuristics

A construction heuristic starts solving a problem from scratch and repeatedly extends the current solution until a complete solution is obtained. It is different from local search in that local search start from a solution and improves it while it starts solving a problem from scratch. Construction heuristics can be used to solve problems like vehicle routing, flow shop scheduling and open shop problem. In the context of vehicle routing the route construction heuristic works towards the solution of the problem by inserting one customer at a time into partial routes until a feasible solution is obtained. The distinguishing feature of construction algorithms is the order in which it selects the customers and the method which it uses to determine where a customer should be inserted. The route construction process can be sequential or parallel. Construction heuristics gradually builds a feasible solution while tracking the current cost of the solution but they lack an improvement phase by itself (Laporte and Semet 2001). A lot of construction heuristics have been proposed over the past 50 years but they are not as popular as they used to be because of the introduction of new techniques such as metaheuristics.

2.6 Evolutionary Algorithms

Evolutionary Algorithms consists of a collection of methods that have been originally developed to solve combinatorial optimization problems. The basis of this discussion is Introduction to Evolutionary Computing, chapter 2 and 10, by Eiben and Smith 2003a. Evolutionary Algorithms fall in the category of "generate and test" algorithms. They are stochastic, population-based algorithms and adapt Darwinian principles to automate problem-solving. The common idea behind all evolutionary algorithms is the same: for a specific population of individuals, the environmental pressure implements a procedure of natural selection, which causes the fitness of the population to increase.

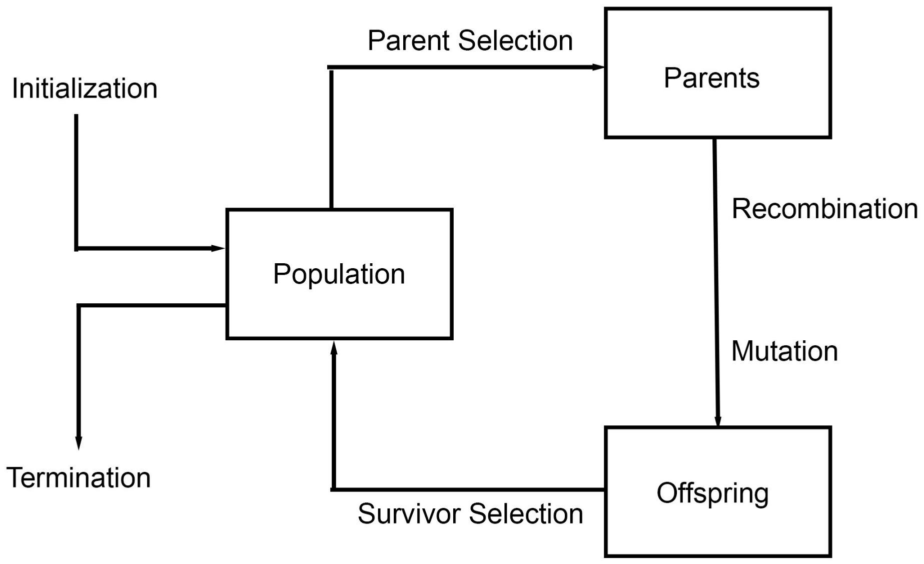

Figure 2: The general scheme of Evolutionary Algorithm

Given a specific quality function that needs to be maximized, we randomly create a set of candidate solutions as shown in figure 2. According to Introduction to Evolutionary Computing, chapter 2, by Eiben and Smith 2003b, we check the fitness of each candidate solution and the ones that are better are selected to seed the next generation by applying recombination/mutation to them. Recombination is applied to two or more parent candidate solutions and results in one or more child solutions. Mutation is applied to one candidate solution and the result is one new candidate solution. The process of recombination and mutation produces a new generation of candidate solutions that compete with the generation of their parents' solutions in fitness and possibly age. The process is repeated till a candidate fulfilling the required fitness is found or the computational limit is reached.

Algorithm 1 General Scheme of Evolutionary Algorithm. (Eiben and Smith 2003c)

```
BEGIN
    INITIALIZE population with random candidate solutions;
    EVALUATE each candidate;
    REPEAT UNTIL (TERMINATION CONDITION is satisfied) DO
        1 SELECT parents;
        2 RECOMBINE pairs of parents;
        3 MUTATE the resulting offspring;
        4 EVALUATE new candidates;
        5 SELECT individuals for the next generation;
    OD
END
```

The best strategy for an evolutionary algorithm is to choose representation that suits our problem. In the next step we need to choose variation operators (recombination and mutation) to suit representation. The selection operators are independent of representation as they only use fitness.

2.6.1 Memetic Algorithms

The combination of evolutionary algorithms with local search heuristic that work with the evolutionary algorithm life cycle is called Memetic Algorithms. Memetic algorithms have been shown to be orders of magnitude faster and more accurate than evolutionary algorithms on some problems. It is a hybrid of evolutionary algorithms. There is a general perception that while pure evolutionary algorithms are quite good at rapidly identifying good areas of the search space but they are not quite efficient in fine-tuning the end solution. The low tuning efficiency is related to the stochastic nature of the variation operators (recombination and mutation). It will be more efficient to incorporate a local search which further increases the quality of solution gained from evolutionary algorithms only, as seen in figure 3.

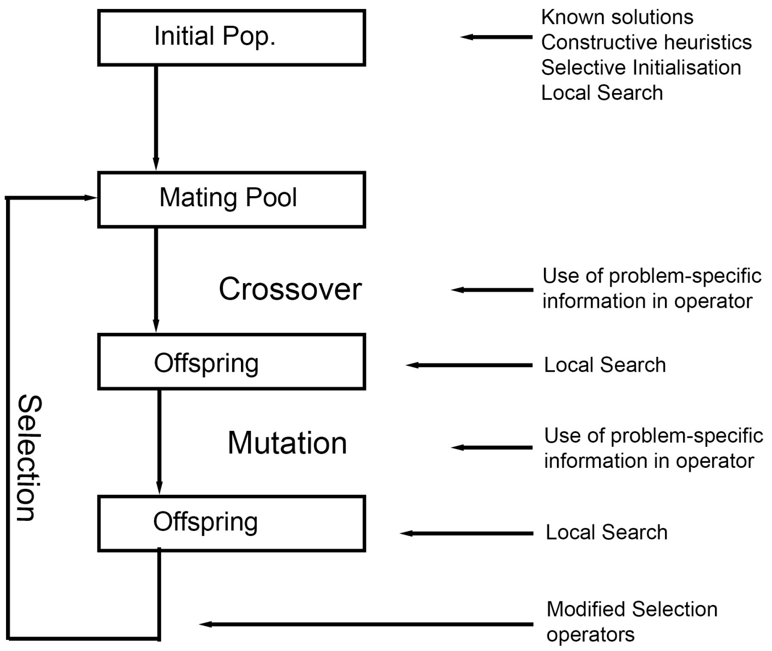

Figure 3: Flowchart of Memetic Algorithm

3 Bin Packing Problems

In this chapter, we introduce the bin packing problem. In order to do that the simple variants such as general container loading and knapsack problem are introduced first. Different approaches for solving the bin packing problem are discussed later on in this chapter. Finally, the two-dimensional bin packing and three-dimensional bin packing problem are introduced and various techniques for solving the problem are discussed.

3.1 Loading Problem

The goal of the loading problem is to load items into a bin or container, minimizing the non-utilized space in the container. According to Dyckhoff 1990, the loading problem is addressed under many names in the literature such as cutting stock, bin or strip packing, trim loss, vector packing, assortment, depletion, dividing, layout, nesting, partitioning, vehicle loading, container loading, pallet loading and knapsack problem, etc. The same paper elaborates on what sort of structure and characteristics are to be considered from a logical perspective while approaching different packing problems. The paper suggests a simple system of 96 (4 x 2 x 3 x 4) combined problems formed by a combination of the four essential characteristics namely dimensions of items, nature of the assignment, sorting of large objects (containers or vehicles) and sorting of small items. The detailed break down of the four characteristics is given in the list below.

1. Dimensionality
   - 1-dimensional such as cutting tubes.
   - 2-dimensional such as cutting glass, cutting wood, 2D container loading.
   - 3-dimensional such as 3D container loading.
   - N-dimensional with N > 3 (the fourth dimension can be time).
2. Nature of assignment
   - All containers (object) and a specification of selected items.
   - A specification of selected containers (object) and all items.

3. Sorting of large objects (containers or vehicles)

    - A single item.
    - Homogeneous figure.
    - Heterogeneous figures.

4. Sorting of small items (to be packed)

    - A small number of items with heterogeneous figures.
    - Numerous items with various heterogeneous figures.
    - Numerous items of various heterogeneous figures (non-corresponding figures).
    - Corresponding figures.

3.2 Knapsack Problem

In the knapsack problem, items with a weight and value are given to put in the knapsack or rucksack. The goal is to fill the knapsack with the most valuable items relative to the items collection. The knapsack problem is important to consider because of its relation to the bin packing problem as discussed in section 3.3. It is a particular case of bin packing where the weight and value of items are the dimensions taken into account. An interesting application of knapsack is container packing or cargo packing. The logistics company has to decide about the transportation requests of the customers. At per unit weight, each request has a weight w_j with a corresponding profit p_j. The capacity of the container is represented by c.

In the formal definition of knapsack problem we are given an item set N, consisting of n items. An item j with a profit p_j and weight w_j and the capacity value c. The objective is to select a subset of items from N such that the total profit of the selected items is maximized and the total weight does not exceed capacity limit c (Knapsack Problem, chapter 1, by Kellerer, Pferschy, and Pisinger 2004a).

The integer formulation of the knapsack is as following:

maximizing

$$\sum_{j=1}^{n} p_j x_j \tag{3.1}$$

under the following constrains

$$\sum_{j=1}^{n} w_j x_j \leq c, \tag{3.2}$$

$$x_{ij} = 0 \; or \; 1, \quad j \in N = \{1, \ldots, n.\}. \tag{3.3}$$

where

$$x_i = \begin{cases} 1 & \text{if item i is selected;} \\ 0 & \text{otherwise} \end{cases}$$

According to Knapsack Problems, chapter 1, by Silvano Martello and Paolo Toth 1990a, the knapsack problem can be interpreted in different contexts. One such instance is an investment problem. An investor has a specific amount of funds c available that he wishes to invest in a beneficial business. She would invest the money w_j, anticipating to get the return p_j for every j decision taken. Each investment is a binary decision and the goal is to maximize the overall return on investment. The knapsack can also be formulated as a cutting problem. Suppose that there is a wooden log that needs to be cut down into small pieces by a saw. These pieces should have a predefined size represented by w_j, and having a selling price p_j. The capacity c of the problem is defined by the wooden log's length.

3.3 Bin Packing Problem

The Bin-Packing Problem can be formulated as follows: there are n number of items and n number of bins. An item j, for which the profit and the weight are represented by p_j and w_j respectively. Each bin's capacity is represented by c. The objective is to assign each item to one bin so that the total weight of items does not exceed the bin capacity c and as few as possible bins are used (Knapsack Problems, chapter 8, by Silvano Martello and Paolo Toth 1990b). This can be formulated as minimize

$$z = \sum_{i=1}^{n} y_i \tag{3.1}$$

subject to

$$\sum_{j=1}^{n} w_j x_{ij} \leq c y_i \quad i \in N = \{1 \ldots n\}, \tag{3.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j \in N, \tag{3.3}$$

$$y_i = 0 \; or \; 1, \quad i \in N, \tag{3.4}$$

$$x_{ij} = 0 \; or \; 1, \quad i \in N, j \in N. \tag{3.5}$$

where

$$y_i = \begin{cases} 1 & \text{if bin i is used;} \\ 0 & \text{if bin i is not used,} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{if item j is assigned to bin i;} \\ 0 & \text{if item j is not assigned to bin i.} \end{cases}$$

The weights w_j are assumed as positive integers. Thus while remaining, in the sphere of generality we will also make the assumption

$$\text{c is a positive integer,} \tag{3.6}$$

$$w_j \leq c \quad for j \in N. \tag{3.7}$$

In case of violation of 3.6, c can be substituted by $\lfloor c \rfloor$. If an item does not fulfill assumption 3.7 i.e. the weight is more than the total capacity of bin, then the instance is trivially infeasible.

The offline algorithm for bin packing works only with complete input data. All workload must be known before the algorithm starts processing the data. In the context of bin packing an offline algorithm has knowledge of the next item in the input sequence required for bin packing. This makes it possible to arrange the items in a particular order before packing the items in the bins. Online bin packing algorithms do not have knowledge of the next item in the input sequence in contrast to the offline algorithm which has knowledge of the next item in the input sequence. Online algorithm packs items in the bin as per the input sequence of incoming items.

In Wäscher, Haußner, and Schumann 2007, bin packing has been further categorized into Single, Multiple, Residual Bin Packing Problem. In single bin packing, identical bins are assigned different items. In multiple bin packing, weakly heterogeneous bins are assigned different items. Whereas in residual bin packing, strongly heterogeneous bins are assigned different items. The name residual depicts the remaining capacity of the bin or remain quantity after cutting a material.

3.4 Heuristics

Heuristic methods have been developed to handle large problem instances of bin packing. Some of the popular heuristic for bin packing are discussed below. Every heuristic is accompanied by an example. Suppose the set of items to be packed as $S = \{3,9,4,1,8,5,2\}$ and the bins' capacity is assumed to be 10.

- **Next-Fit (NF) Algorithm:** In Next Fit the placement of items is carried in the sequence of their arrival. Subject to the condition that it fits, the next item is placed into the current bin. Otherwise, a new bin is started. The result is exhibited in figure 4. The result of the Next Fit algorithm according to our example is five bins which is evidently wasteful but the same result may be accepted if there is no information about the free space in previous bins.

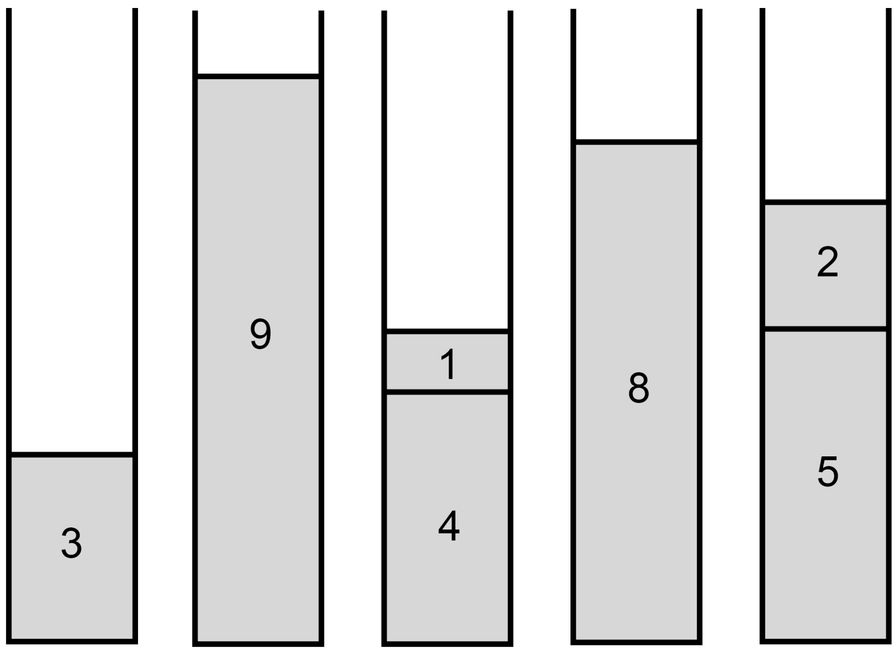

Figure 4: Applying Next Fit Algorithm for Packing

- **First-Fit (FF) Algorithm:** In First Fit the placement of items is carried in the sequence of their arrival. The next item is placed into the lowest numbered bin in which it fits. A new bin is started if the item does not fit into any of the presently opened bin. The result is exhibited in figure 5 showing the packing in four bins. This algorithm is dependent on keeping the previous bins in memory.

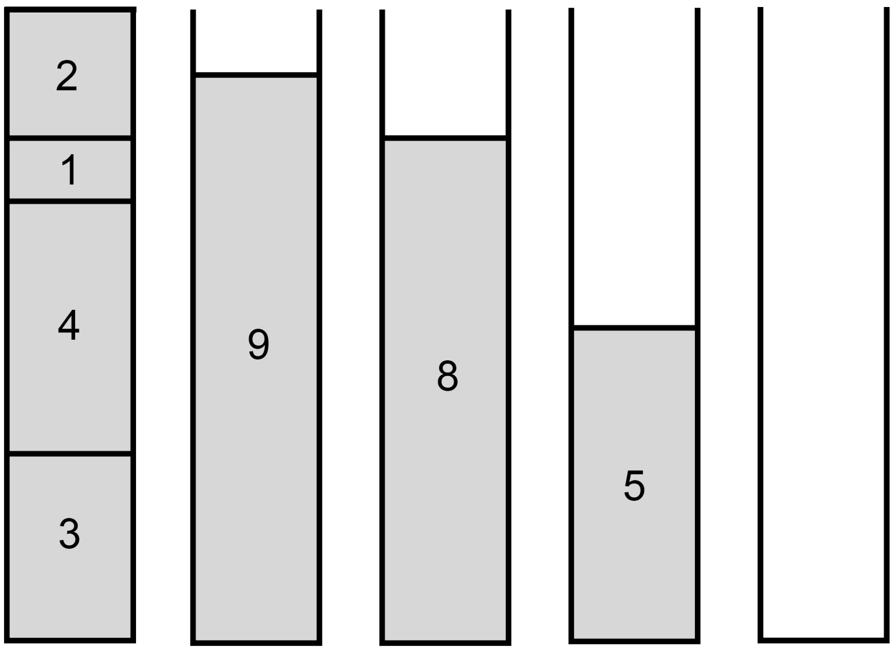

Figure 5: Applying First Fit Algorithm for Packing

- **Best-Fit (BF) Algorithm:** In Best Fit the placement of items is carried in the sequence of their arrival. The next item is placed into a bin such that the minimum space will be left over after its placement in that bin. A new bin is started when the item doesn't fit in any of the available bins. The result of this algorithm is exhibited in figure 6. In the case of the Best Fit algorithm, the result is four bins as well and it is dependent on keeping track of previous bins.

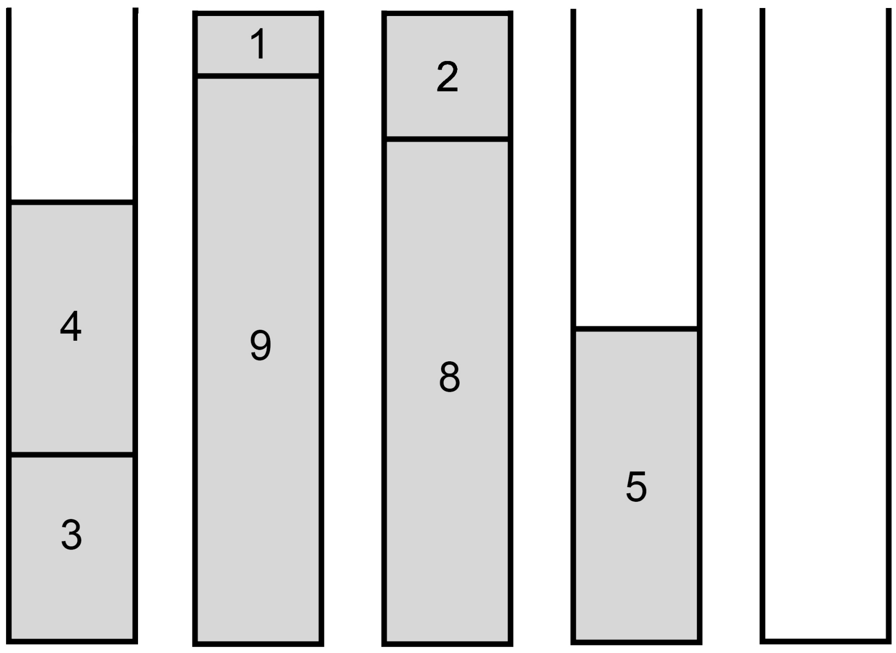

Figure 6: Applying Best Fit Algorithm for Packing

- **Worst-Fit (WF) Algorithm:** In Worst Fit the items are placed in the order of their arrival. The next item is placed into a bin such that the most room will be left over after the placement of that item in the same bin. A new bin is started, if the item doesn't fit into any of the available bins. The result is exhibited in figure 7 and amounts to five bins.

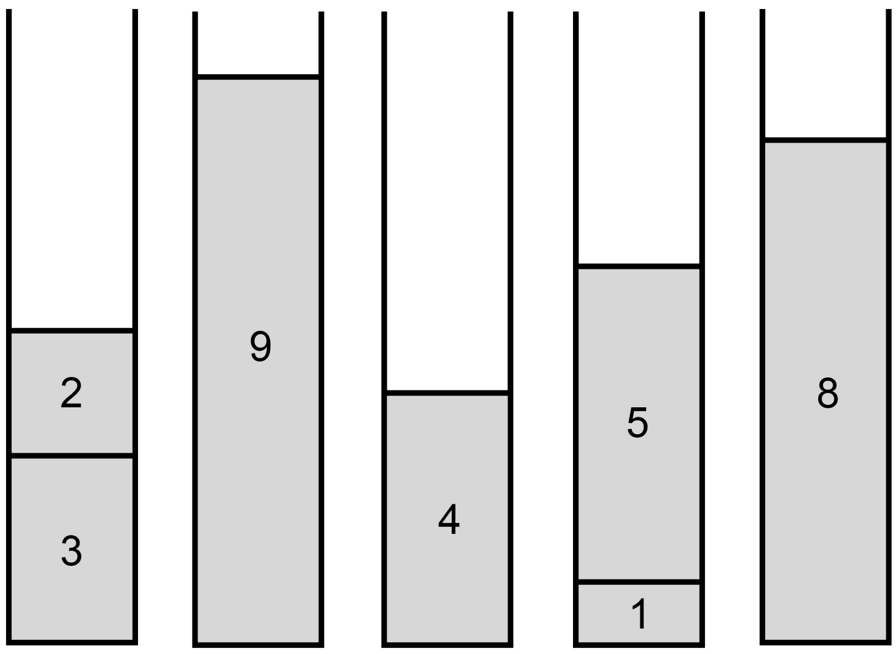

Figure 7: Applying Worst Fit Algorithm for Packing

- **First-Fit Decreasing and Best-Fit Decreasing (FFD-BFD) Algorithm:** In First Fit Decreasing and Best Fit Decreasing the items are first sorted in descending order. In this instance, both the First Fit or Best Fit algorithm may be applied for the reason that they yield equivalent solutions. The use of these algorithms is exhibited in figure 8 and amounts to four bins. It is the best result so far.

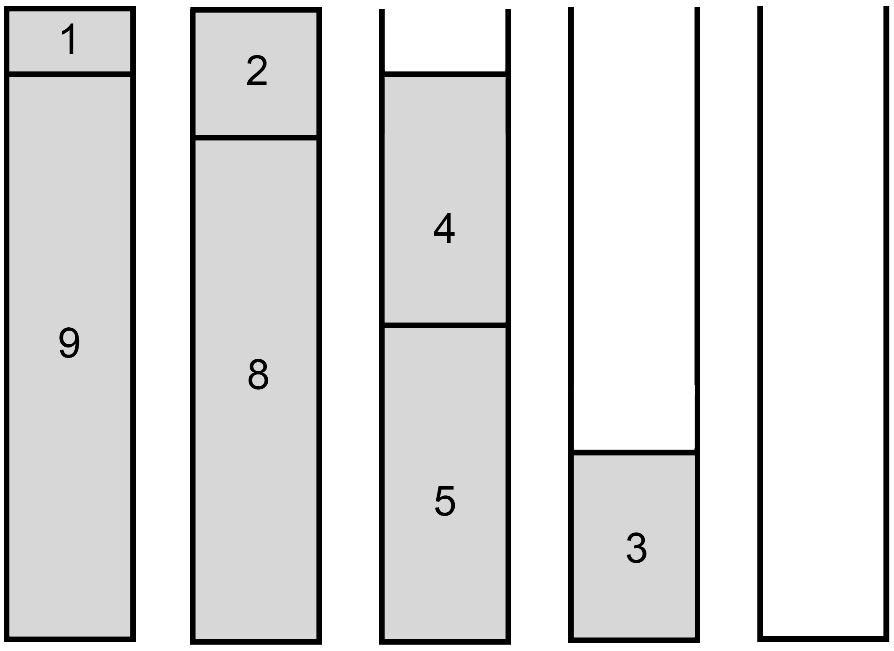

Figure 8: Applying First-Best Fit Decreasing Algorithm for Packing

3.5 Exact Algorithms

A brief outline of exact algorithms is discussed based on Knapsack Problems, chapter 8, by Silvano Martello and Paolo Toth 1990b. Bin packing is NP-hard problem and the exact solution to bin packing problem takes exponential time. Eilon and Christofides 1971 developed a heuristic algorithm to solve the problem with different objectives, that is to minimize the number of bins; minimize the un-accommodated number of items; and a combination of both. This algorithm basis its approach on best fit decreasing branching. For a given decision node, the initialization of b bins is assumed, let the current residual capacities of the bins sorted in the ascending order be exhibited by $(\bar{c}_{i_1} \ldots \bar{c}_{i_b})$, and $\bar{c}_{i_{b+1}} \equiv c_{b+1} = c$ be the capacity of the next bin which is still uninitialized: the free item j^* weighing the largest is assigned by the branching phase to bins $i_s, \ldots, i_b, i_{b+1}$, where $s = min\{h : 1 \leq h \leq b+1\ ,\ \bar{c}_{i_b} + w_{j^*} \leq c\}$. This algorithm is applicable to small scale instances only.

A branch and bound algorithm for a generalization of bin packing problem was presented in

Hung and Brown 1978. In this algorithm, bins with different capacities can be accommodated. The branching strategy in this algorithm is based on the characterization of equivalent assignments which is used to develop a set of rules for generating non-redundant assignments. This approach reduces the number of explored decision nodes.

Another algorithm named MTP has been proposed by S Martello and P Toth 1989. It is based on a first-fit decreasing branching strategy. The items are classified on the basis of the descending order of their weight from the start. The bins are indexed in order of their initialization. The first largest free item is allocated at each decision node, in turn, to the initialized bins that are feasible in ascending order of their index, and to a new bin. In the upcoming steps, procedures are called on a node to fathom it and minimize the current problem. When it is not possible to fathom the node then approximate algorithms such as First Fit Decreasing, Best Fit Decreasing and Worst Fit Decreasing are applied for the improvement of the current problem. A backtracking step happens in the form of removing the current item j^* from its current bin i^* and it is allocated to the next feasible bin.

3.6 Approximate Algorithms

Bin packing is an NP-hard problem and the exact solution to bin packing problem takes exponential time. This increases the difficulty of finding the exact solution. It would need more resources and time even for a small instance of bin packing problem. In order to handle this situation in a better way, approximation algorithms are used. In many instances it is not necessary to find the optimal solution for the bin packing problem. An approximate solution that is close to an optimal solution and computed with the help of reasonable resources and time is considered a good solution.

A brief outline of approximate algorithms is discussed based on Knapsack Problems, chapter 8, by Silvano Martello and Paolo Toth 1990b. Next Fit (NF) is a simpler approximate approach to bin packing as discussed in section 3.4. The complexity of the Next Fit (NF) is $O(n)$. It can be proved for any instance I of the bin packing problem, the solution value $NF(I)$ provided by the algorithm fulfills the bound

$$NF(I) \leq 2\, z^*(I), \tag{3.1}$$

Here $z^*(I)$ denotes the optimal value of the solution. Additionally, there are instances having worst-case performance ratio $NF(I)/z^*(I)$ close to 2 i.e. $r(NF) = 2$. The worst-case performance ratio is defined as the smallest real number $r(A)$ such that

$$\frac{z^A(I)}{z^*(I)} \leq r(A) \quad \text{for all instances I,}$$

Another algorithm is called First Fit (FF) as discussed in section 3.4. It has been proved in Johnson et al. 1974 that

$$FF(I) \leq \frac{17}{10}\, z^*(I) + 2 \tag{3.2}$$

for all the instances I of the bin packing problem, and also there are such instances I, with arbitrarily huge $z^*(I)$, for which

$$FF(I) \leq \frac{17}{10}\, z^*(I) - 8 \tag{3.3}$$

As we can see that a constant term is used in inequality 3.2, so the worst-case performance does not provide full information on the worst-case behavior. The asymptotic worst-case performance ratio is generally applied in place of worst case performance for bin packing algorithms. For an approximate algorithm A. This is defined as the minimum real number $r^\infty(A)$ such that, for some positive integer k,

$$\frac{z^A(I)}{z^*(I)} \leq r^\infty(A) \quad \text{for all instances I satisfying } \; z^*(I) \geq k;$$

it can be clearly seen from inequalities 3.2 and 3.3 that $r^\infty(FF) = \frac{17}{10}$.

The next algorithm is Best Fit (BF) as discussed in section 3.4 . Johnson et al. 1974 proved that best fit has the same worst-case bounds as first fit (as seen from inequalities 3.2 and 3.3), hence $r^{\infty}(BF) = \frac{17}{10}$. The time complexity of the first fit and best fit is $O(n \log n)$.

The algorithms First Fit Decreasing (FFD) and Best Fit Decreasing (BFD) are discussed in section 3.4. The worst-case analysis of first-fit decreasing and best fit decreasing are done in Johnson et al. 1974 extending the below inequality from Johnson 1973 for all instances I.

$$FFD(I) \leq \frac{11}{9} z^{*}(I) + 4 \tag{3.4}$$

The table 1(selected data) taken from Coffman Jr, Garey, and Johnson 1984, summarizes the rest of the results. In this table the last three columns give the value r_{α}^{∞} of the asymptotic worst case performance ratio of the algorithms when applied to instances satisfying $min_{1 \leq j \leq n}\{w_j\} \leq \alpha c$

Algorithm	Time Complexity	r^{∞}	$r_{\frac{1}{2}}^{\infty}$	$r_{\frac{1}{3}}^{\infty}$	$r_{\frac{1}{4}}^{\infty}$
NF	$O(n)$	2.000	2.000	1.500	1.333 ...
FF	$O(n \log n)$	1.700	1.500	1.333 ...	1.250
BF	$O(n \log n)$	1.700	1.500	1.333 ...	1.250
FFD	$O(n \log n)$	1.222...	1.183...	1.183 ...	1.150
BFD	$O(n \log n)$	1.222...	1.183...	1.183 ...	1.150

Table 1: Asymptotic worst-case performance ratios of bin-packing algorithms (Coffman Jr, Garey, and Johnson 1984).

3.7 Two Dimensional Bin Packing Problem (2DBPP)

In 2-D bin packing the objective is to pack a set of unique rectangular objects into rectangular bin avoiding any overlap. The number of rectangular bins is assumed to be unlimited. An item is classified by its height and width. This discussion of various algorithms for 2DBPP is based on Lodi, Martello, and Monaci 2002 and Mahvash-Mohammadi 2014.

Gilmore and Gomory 1963 extended their 1D-BPP approach to model 2D-BPP for the first time. In this paper, an earlier method for stock cutting is extended and adapted to the specific full-scale paper trim problem. Fekete and Schepers 2000 presents a new approach for modeling packings, using a graph-theoretical characterization of feasible packings. The characterization allows it to deal with classes of packings that share a specific combinatorial structure, instead of considering one packing at a time. Pisinger and Sigurd 2007 gives an exact algorithm based on the Dantzig-Wolfe decomposition where the master problem deals with the production constraints on the rectangles while the subproblem deals with the packing of rectangles into a single bin. Martello and Vigo 1998 addressed the two dimensional finite bin packing problem through an exact algorithm. Caprara and Monaci 2009 provided an exact algorithm for geometric problems in which rectangles have to be packed into (identical) squares. Most of the exact approaches use a branch and bound method.

Coffman et al. 1980 provides various greedy algorithms for 2DBPP. A greedy algorithm makes the locally optimal choice at each node in a search space with the goal of finding the global optimum. They commonly employ the concept of different layers. The layers start at the bottom of the bin or container. The next layer is a horizontal line drawn parallel to the top of the highest item located on the previous layer. This way the bin is filled layer by layer. In most of the greedy algorithms for 2DBPP, items are sorted with respect to their height in non-increasing order and then iteratively packed according to the following schemes:

- **Next Fit Decreasing Height (NFDH):** Each new item is packed in the current layer starting from the bottom left. If the item cannot be packed on the current layer then the layer is closed and a new current layer is created on top of the closed layer.
- **First Fit Decreasing Height (FFDH):** Each new item is packed in the first existing layer starting from the bottom left. If the item cannot be packed in any of the existing layer then a new layer is created on top of the existing layer.
- **Best Fit Decreasing Height (BFDH):** Each new item is packed on the fitting layer with the minimum horizontal space remaining. If there is no fitting layer available, a new one is created.

3.8 Three Dimensional Bin Packing Problem (3DBPP)

In 3DBPP, we are given a set of n rectangular shaped items, each having dimensions of width, height and depth. The volume capacity of the bin is also expressed as a 3D loading space having dimensions of width, height and depth. The number of bins is assumed to be unlimited. The 3DBPP consists of orthogonally packing all the items in a minimum number of bins. Any two items placed in the bin should not overlap with each other. This discussion of various algorithms for 3DBPP is based on Zhao et al. 2016 and Mahvash-Mohammadi 2014. Chen, Lee, and Shen 1995 describes the packing of cartons of different sizes into containers. The 3DBPP problem is expressed as a zero-one mixed-integer programming model with weight distribution and orientation constraints. To verify the above model and find out its computational time, an instance problem of three unequal-sized containers and six non-uniform cartons was solved with LINGO (commercial mathematical programming package) solver.

The first exact algorithm for 3DBPP was designed by Martello, Pisinger, and Vigo 2000. It was a two-level branch and bound algorithm. A first-level search assigns boxes to bins. At each node of the first level search tree, a second-level branch and bound method is used to ascertain the feasible ways in which the items can be packed employing concept of corner points. Corner points refer to such points at which new items are adjusted in the container's remaining empty space, in a given partial packing. Corner points reduce the number of partial solutions explored. There are two heuristic algorithms that are applied at every root node in the search tree. Considering the bins' depth, the first algorithm constructs a number of layers and accumulates them into bins by using the solution of a one-dimensional bin packing. The second algorithm reduces the empty space of the bins to a minimum by filling each bin to the fullest. This exact approach can solve real-life problems of up to 20 boxes optimally within a feasible computational limit.

Den Boef et al. 2005 argued that since Martello, Pisinger, and Vigo 2000 cannot generate all feasible orthogonal patterns which might make it fail to get an optimal solution. In response, Martello et al. 2007 has improved its approach from Martello, Pisinger, and Vigo 2000 by using a combination of the original enumerative method and a new constraint-based programming. The new algorithms can solve moderate-sized instances to get an optimal solution.

Faroe, Pisinger, and Zachariasen 2003 developed a Guided Local Search (GLS) heuristic method for three-dimensional bin packing. A greedy heuristic method is used by the algorithm to obtain the upper bound on the number of bins. It tries to minimize the upper bound by finding a feasible way in which the boxes can be packed in each iteration. The process ends when a specified computational limit of time or precomputed lower bound is arrived at. This approach can solve instances with up to 200 boxes and provides results that are better than other heuristics or exact algorithms proposed before it.

Lodi, Martello, and Vigo 2002 introduced a Tabu Search framework exploiting a new constructive heuristic for the evaluation of the neighborhood. This approach uses Height first Area second (HA) layers. The first layer is marked as the base of a bin. The base of the second layer is marked at the height of the first layer's tallest item, and so on. HA selects the best solutions through the following two methods:

1. The items are sorted into clusters according to non-increasing height, and a solution based on layered strip packing is determined. The obtained layers are then combined into finite bins through a 1DBPP algorithm.
2. The items are sorted again according to the non-increasing area of their base and allocated again to the current layers, possibly altering the layer heights. The layers are then packed into bins using 1DBPP.

Satisfactory solutions are produced by combining tabu search with height first area second heuristic as compared to exact and heuristic methods presented by Martello, Pisinger, and Vigo 2000. In about 30% of instances, the tabu search and Faroe, Pisinger, and Zachariasen 2003 gives exactly the same average solution. In some instances, tabu search is more efficient than guided local search, and vice versa.

For 3DBPP, Parreño et al. 2010 suggested a Greedy Randomized Adaptive Search Procedure (GRASP). To obtain an efficient solution this algorithm uses a combination of a constructive procedure and an improvement phase at each iteration. A maximal-space heuristic is applied in the constructive phase. For the purpose of improving the solution, in a Variable Neighborhood Descent (VND) structure, different moves are employed.

4 Vehicle Routing Problems

The family of vehicle routing problems can be generically defined (Vehicle Routing, chapter 1, by Irnich, Toth, and Vigo 2014) as follows; Where there is a given fleet of vehicles and transportation requests, the aim is to fulfill those requests using the available fleet at minimum expenditure by deciding a set of routes. Particularly, emphasis is put on the provision of feasible services to all the routes by determining a suitable vehicle and sequence of customers for a route.

The original vehicle routing problem was first analyzed in Dantzig and Ramser 1959. The problem was about distributing gasoline to different service stations around the city. The authors of that paper wanted to find out the optimum routes for the oil delivery trucks. Each truck contains a fuel tanker. The goal is to use the least possible trucks, that are driving the least possible distance to deliver oil to all service stations. Another constraint is that the truck should not run out of fuel while covering its route. Even after all those years of research, large-scale instances based on real-life scenarios or complex versions of this problem are still challenging for researchers. This chapter provides the background, description and various methods to solve the vehicle routing problem.

4.1 The Problem of the Traveling Salesman

The traveling salesman problem is a touring problem in which each city must be exactly visited once on a tour. The goal is to find the shortest possible tour while satisfying the constraints (Artificial Intelligence, chapter 3, by Russell and Norvig 2003a). The problem is NP-hard but researchers have given it a lot of emphasis due to its application in real-life problems like automatic circuit board drill, stocking machines on shop floor and vehicle routing problems.

According to Ropke 2005, there are different variants of the traveling salesman problem. In symmetric variant for all cities i.e. x and y, the distance between the cities is the same from both directions. In asymmetric variants for all cities i.e. x and y, the distance between the cities is not the same from both directions. In Euclidean variants, the cities must be located in $\mathbb{R}^d$and the distance between the cities is the euclidean distance.

The traveling salesman problem is defined on a directed graph G consisting of vertices V and arcs A. The various customer nodes are exhibited by the set, V, of vertices i.e.$\{1,\ldots,n\}$, while the set of directed edges is exhibited by the set, A, of arcs. Figure 9 shows an example of a traveling salesman problem with a directed graph.

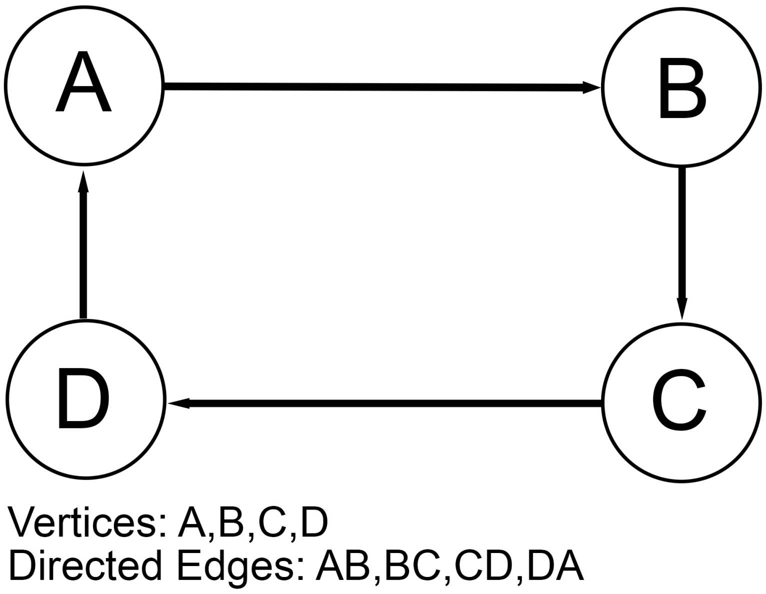

Figure 9: Traveling Salesman Directed Graph

Each $(i,j) \in A$ is assigned a distance or cost c_{ij}. A binary decision variable x_{ij} is set to one, subject to the condition that arc (i,j) is used in the solution. The problem can be expressed as

$$\sum_{i\in V}\sum_{j\in V\setminus\{i\}} c_{ij}x_{ij} \tag{4.1}$$

minimizing the above, subject to

$$\sum_{j\in V\setminus\{i\}} x_{ij} = 1 \quad \forall i \in V, \tag{4.2}$$

$$\sum_{i\in V\setminus\{j\}} x_{ij} = 1 \quad \forall j \in V, \tag{4.3}$$

$$\sum_{i\in S}\sum_{j\in V\setminus S} x_{ij} \geq 1 \quad \forall S \subset V, \tag{4.4}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A. \tag{4.5}$$

The objective function 4.1 minimizes the arc cost, and to make sure that each node has only one incoming and outgoing arc we have the constraints 4.2 and 4.3, while 4.4 prohibits the possibility of sub-tours. Sub tours are multiple tours rather than one big tour through all the points.

There is a great amount of research present on the traveling salesperson problem. This historic origin of the traveling salesman problem is discussed in The Traveling Salesman Problem, chapter 1, by Lawler 1985 and Schrijver 2005. The traveling salesman problem is the parent problem of the vehicle routing problem. It is analogous to a vehicle routing problem with a fleet of one vehicle.

4.2 The Problem of multiple Traveling Salesman

This problem is a generalized version of the traveling salesman problem that consists of multiple salesman. The problem consists of x cities, m salesman and a depot. Every city must have only one visit on any of the m tours. Every m tour starts and ends at the depot. No tour is allowed to be empty. The objective of the m-TSP is to minimize the total tour cost by determining a feasible tour for each salesman.

A number of variants of m-TSP are briefly defined below.

Multiple depots In multiple depot m-tsp a salesman can return to his initial starting depot or another depot which is different than his initial depot. The initial number of salesman on each depot must remain the same after all the trips are finished.

Number of salesman The number of salesman in the problem may be a fixed number *m*, or it might be determined by the solution with an upper bound m.

Cost In a non-fixed salesman scenario, a cost is attached to the usage of each salesman. This motivates the minimization of total salesman cost, in addition to minimizing the total tour cost.

Time Frame Each customer can be only visited within an allowed time window. This variant is quite useful in vehicle routing with time windows.

Constraints Constraints can be applied to the number of nodes visited by each salesman, maximum or minimum distance a salesman can travel or any other constraints.

4.3 Problem Notions for Vehicle Routing

As described in section 2.3, vehicle routing problems focus on minimizing a cost function. The cost function is generally taken as total distance covered by all the vehicles to interact with all customers and return to their initial position. Vehicle routing also includes delivery or pick up of goods from customers using the given set of vehicles. The chosen solution at the end of solving this problem should give minimal cost and satisfy all customers while complying with additional side constraints. Additional constraints can be limiting the number of customers that each vehicle can visit, the time window in which each customer should be visited, the order in which a customer should be visited, and so on. Some of the elements encountered in vehicle routing problems are described below:

Customers Customers are the locations that are visited for the purpose of picking up or delivering goods. They represent real geographical locations on the map. They can be alternatively called vertices or nodes.

Vehicles Vehicles are the objects that are used to transfer the goods to or from customers. There can be different attributes for the vehicle in various real-life scenarios. The vehicle can have one or more compartments. In case of more than one compartment, they can be homogeneous or heterogeneous in terms of capacity.

Depot Depot is a real geographical location that is used to store goods. Generally in vehicle routing problems, depot represents the home for vehicles. Vehicle starts delivery on a route from the depot and finishes its route on the depot. There can also be multi depots in a vehicle routing problem.

Road network Road network is used by vehicles to travel for the purpose of delivering or picking up goods from the customers. In theory the graph exhibits a road network, where the vertices represent the customer and the depots. The edges in the graph represent the road segment between different customers and depots. We give a weight to edges between vertices and depots. This weight is proportionate to real-life length of the road segment connecting the respective entities. It could also be driving time, expected fuel consumption, and other costs as well.

Routes Routes comprises sequential locations that are visited by one vehicle on a single trip. A trip can be thought of as starting from the depot and returning back to the depot.

Fleet Fleet comprises of a set of vehicles used to transfer goods between customers and depot. There are two types of fleets i.e. homogeneous fleet and heterogeneous fleet. In a homogeneous fleet, all the vehicles have the same capacity and performance while in the heterogeneous fleet, the performance and capacity in a set of vehicles can differ.

Objective Objective of the vehicle routing problems is minimization or maximization of some quantity. Generally the cost function of this operation is minimized. The cost function is generally taken as total distance covered by all the vehicles, to interact with all customers, and return to their initial position.

Solution Solution is a plan that gives minimal cost and satisfies all customers while complying with additional side constraints. It is generally a set of routes in which each route is traveled by one vehicle from the given vehicle set.

Constraints Constraints are specific conditions that are required to be satisfied while giving service to all the customers in a vehicle routing problem. Constraints can be related to visiting a group of customers in a sequence or in specific time windows in which it is possible to load and unload items from customers, and so on.

4.4 The Capacitated VRP

The Capacitated Vehicle Routing Problem (CVRP) is the most basic category among different variants of the vehicle routing problems.

4.4.1 Problem Description

The capacitated vehicle routing problem is about delivering service to a set consisting of customers and their specific demands by using a set consisting of a specific number of vehicles. Each vehicle is considered to have a limited capacity. All vehicles start and end their journey at the depot. The goal is to divide the customers into as less routes as possible, hence minimizing the distance. It should be kept in focus that the truck capacity for a route should not be excessive then the threshold limit of the truck. The problem is described in the section with relevant notations. The basis of this discussion is Vehicle Routing, chapter 1, by Irnich, Toth, and Vigo 2014 and Massen 2013.

The problem is illustrated on a graph $G = (V,A)$ having the characteristics mentioned in section 4.1 such that the outgoing and incoming arcs are exhibited by δ_i^- and δ_i^+ respectively where i is part of vertices V. Vertexes $i = 1,\dots,n$ represent the customers, while the depot is represented by vertex 0. c_{ij} of an arc $(i, j) \in A$ is the equivalent weight of the distance between such location which matches vertex i and j. The distances are assumed to be symmetric and to comply with the triangle inequality. There is an associated demand i.e. q_i for each customer. For executing the visits to the customers, K a fleet of homogeneous vehicles is available. Each of which has a limited capacity Q and a maximum of one route to execute. The starting and ending point of each route is the depot. As the depot vertex comprises of first and last vertexes of a route, a tuple (S,σ) can be used to depict a route r such that $r.\sigma = \langle e_1,\dots,e_m\rangle$ $(\cap_{i=1}^{m}\{e_i\} = \varnothing)$ is a sequence of customers $(\{e_1,\dots,e_m\} \subseteq V \setminus \{0\})$ and $r.S = \{e_1,\dots,e_m\}$. Furthermore $r[s]$ $(1 \le s \le |r.S|)$ shows the s^{th} position in the route r where the customers are visited. For any route r the depot is represented by $r[0]$ and $r[|r.S| + 1]$.

The intention is to make a solution $Sol = \{r_1 \dots r_m\}$ containing a set of routes fulfilling the following conditions:

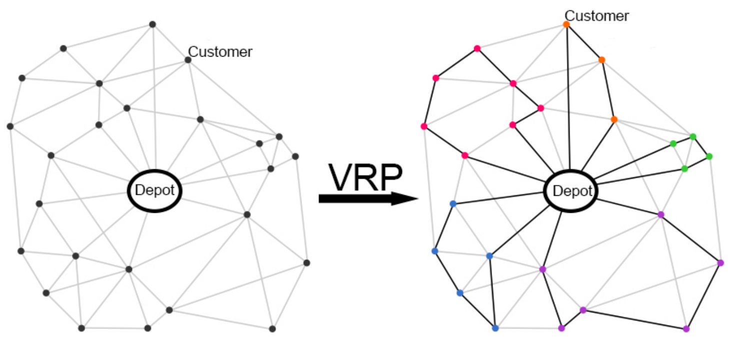

Figure 10: Vehicle Routing Problem (based on *Vehicle Routing Problem* 2013)

1. The solution should not surpass the number of vehicles than available.
2. To make sure that each customer is visited exactly once for delivering goods.
3. The total volume of items to be packed for a route should be less than or equal to vehicle capacity Q.
4. Minimize the total distance.

A feasible solution will be one complying with constraints 1-3. If a solution does not comply with the constraints partially or totally, it is called infeasible. A solution that gives the least distance while satisfying constraints is considered to be a higher quality solution. A solution is optimal if there is no solution that has a higher quality than the solution under consideration. Figure 10 depicts an example of solving the vehicle routing problem.

4.4.2 Capacitated VRP Variants

Several variants of CVRP are discussed in The Vehicle Routing Problem by Golden, Raghavan, and Wasil 2008, Vehicle Routing by Toth et al. 2014 and Massen 2013. The types of modifications that give birth to a new CVRP variant along with examples are given below.

1. Changes to routes structure

- **Vehicle Routing Problem with Multi-depot**
  As opposed to a single depot, starting and ending point of the vehicles are in different depots, as discussed in Renaud, Laporte, and Boctor 1996.
- **Capacitated Vehicle Routing Problem With Split Delivery**
  Dror and Trudeau 1990 discusses the splitting of an order that can't be delivered in the current vehicle capacity into multiple smaller orders that are delivered by different vehicles.
- **Capacitated Vehicle Routing Problem With Multiple Trips**
  As discussed in Taillard, Laporte, and Gendreau 1996, in the VRP with multiple trips, several routes may be executed by a single vehicle.

2. Changes to objective function

- **Vehicle Routing Problem with Profits**
  Each customer has an associated profit for the provision of service. With limited fleet size it may be impossible to handle a lot of customers. In the above situation it is better if the service is provided to a smaller number of customers while maximizing profit (Archetti, Speranza, and Vigo 2014).
- **Vehicle Routing Problem with MinMax objective**
  Rather than minimizing the total distance, a min-max objective may be applied, e.g., for the purpose of minimizing the length, duration or workload of the longest route (Golden, Laporte, and Taillard 1997).
- **Vehicle Routing Problem with the objective of minimizing the vehicle fleet**
  The primary objective is to minimize the number of vehicles in the fleet. This is due to the fact that vehicles and drivers form a bigger portion of the service cost for the route trips. A common order of optimizing is to first minimize the number of vehicles and followed by a secondary objective such as the minimization of the total distance of the trips (Bräysy and Gendreau 2005).

3. Putting additional constraints while providing service to customer

- **Capacitated Vehicle Routing Problem with Time Windows**
  Each customer and the depot have a specific time window associated with it. This is CVRP with scheduling constraint i.e. requiring the time of the travel, service

and waiting times to be considered together with time-window constraints. The service to each customer should be provided within the customer's time window. It is possible for a vehicle to wait if they arrive before the customer time slot is started. The service time for each customer is predefined. The return of vehicle to depot must also be before the end of the depot's time window (Cordeau et al. 2001).

- **Capacitated Vehicle Routing Problem with Pick-up and Delivery**
  Pickup-and-delivery problems are vehicle routing problems that consist of transportation requests (of goods or people) from one particular location to another. The pick-up location should be visited before the delivery location for a respective customer request. The loading space of the vehicle will not fluctuate during the execution of the route (Desaulniers et al. 2001).
- **Capacitated Vehicle Routing Problem with Loading Constraints**
  When 2D or 3D quantities describe the pallet and the cargo compartments, it can result in complex loading constraints. Iori, Salazar-González, and Vigo 2007 discusses 2-D loading constraints in which rectangular items are loaded in rectangular compartments. There might be orientation constraint, sequential loading constraint and capacity constraint. Gendreau et al. 2006 discusses stability constraint, fragility constraint, and ease of unloading of boxes at the destination in three-dimensional loading.

4.5 Construction Heuristics

A constructive heuristic starts solving a problem from scratch. The goal is to build a good quality solution to the problem. It constructs a set of routes for the VRP problem. Construction heuristics gradually builds a feasible solution while tracking the current cost of the solution but they lack an improvement phase by itself (Laporte and Semet 2001). This is done by repeatedly extending the current solution (initially empty) until a complete solution is obtained. The route construction process can be sequential or parallel. In the process of constructing routes, the distance of the solution has to be kept minimum. Route construction heuristic work towards the solution of the problem by inserting one customer at a time into partial routes till the construction of a feasible solution. The basis of discussion for this topic is Vehicle Routing, chapter 4, by Laporte, Ropke, and Vidal 2014, Laporte and Semet 2001 and Massen 2013. The notations are mostly based on Massen 2013. Some of the commonly encountered construction heuristics for CVRP are given below.

4.5.1 Savings Heuristic

The Clarke and Wright algorithm (Clarke and Wright 1964) is a popular heuristic based on the notion of saving. In the paper, a sequential and parallel version is proposed. We will discuss the parallel version as it appears to be better performing according to Laporte and Semet 2001. At each iteration, the merger of the feasible route is implemented in order to obtain the largest saving to the point where the merger is not feasible anymore, as shown in figure 11. While being simple, this algorithm has got the advantages of being intuitive, fast and easily implementable. It can also be used to generate an initial solution for other algorithms to work on.

When two routes $r_1 = (0,\dots,i,0)$ and $r_2 = (0,j,\dots,0)$ can be feasibly merged into a single route $(0,\dots,i,j,\dots,0)$, a distance saving $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ is generated. The heuristic then allots every single customer a specified route such that, $Sol = \{r_1,\dots,r_n\}(n = |V\backslash\{0\}|)$ and $r_i.S = \{i\}(1 \leq i \leq n)$. As a next step, the routes in Sol are being merged. The ordered pair (i,j) such that $(i,j \in V\backslash\{0\}\,, i \neq j)$ is maximizing s_{ij}, and such that $\exists r_a, r_b \in Sol$ having $first(r_b) = j$ and $last(r_a) = i$ and having $demand(r_a) + demand(r_b) \leq Q$ is decided at each iteration. The two routes are then merged together. Hence, a new route $r^{'} = r_a \times r_b$ is formed which replaces r_a and r_b in Sol. Thus the new solution with a reduced set of routes is given as $Sol = (Sol\backslash\{r_a, r_b\}) \cup \{r^{'}\}$.

The routes are continuously merged by the heuristic till it is not possible anymore. The remaining set of routes, which is obtained after execution of the heuristic, is our final solution. We can use this solution if we have the freedom of using vehicles as per the final routes of our solution. If available vehicles are fixed and less than the number of routes formed, then the solution is infeasible.

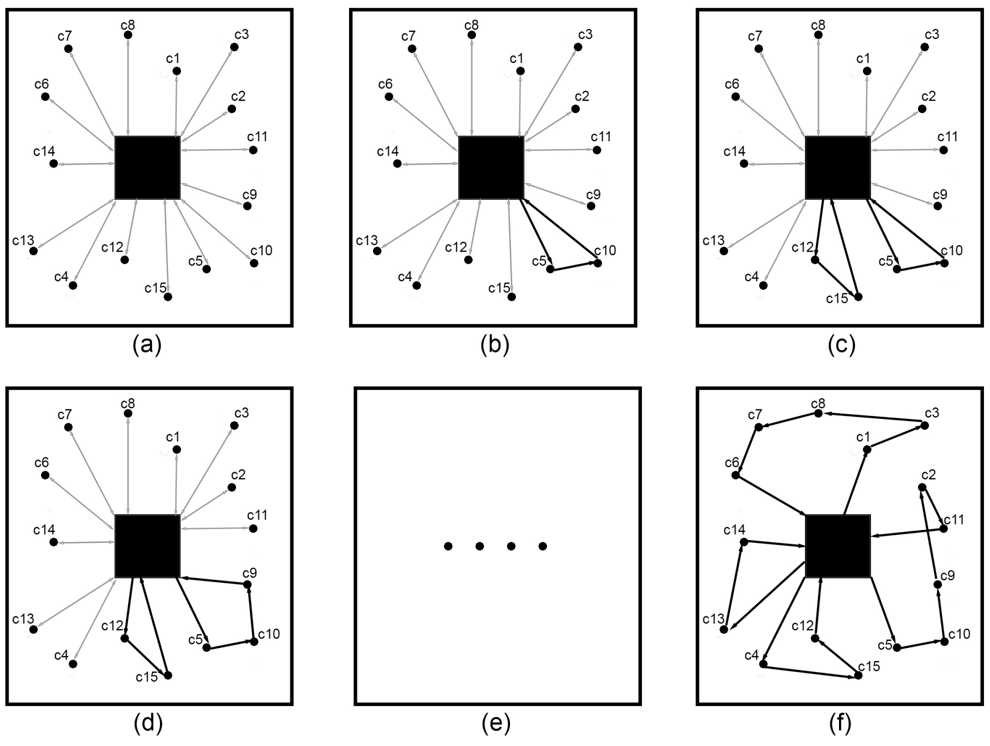

Figure 11: Running the Savings Heuristic

4.5.2 Insertion Heuristic

Insertion heuristic develops a solution that consists of filling the open routes by the insertion of one customer after the other. An open route is a route where customers can still be inserted. A closed route is used to denote routes where customers can't be inserted anymore. Insertion heuristics are then divided into sequential insertion and parallel insertion. Sequential insertion heuristics build a single route at a time while parallel insertion heuristics construct multiple or all routes in parallel. Sequential insertion discussed in R. H. Mole 1976 is shown in figure 12.

Insertion heuristic starts solving a problem from scratch. It constructs a set of routes by making a decision with respect to the customer that is to be inserted, and the position in the route of such insertion at each iteration. The family of insertion heuristics is differentiated on the basis of location and sequence of customer insertion in routes. One example of insertion heuristic could be the insertion of such a customer that keeps the overall cost as low as possible. Another heuristic could be to keep one route open at a time and choose the nearest

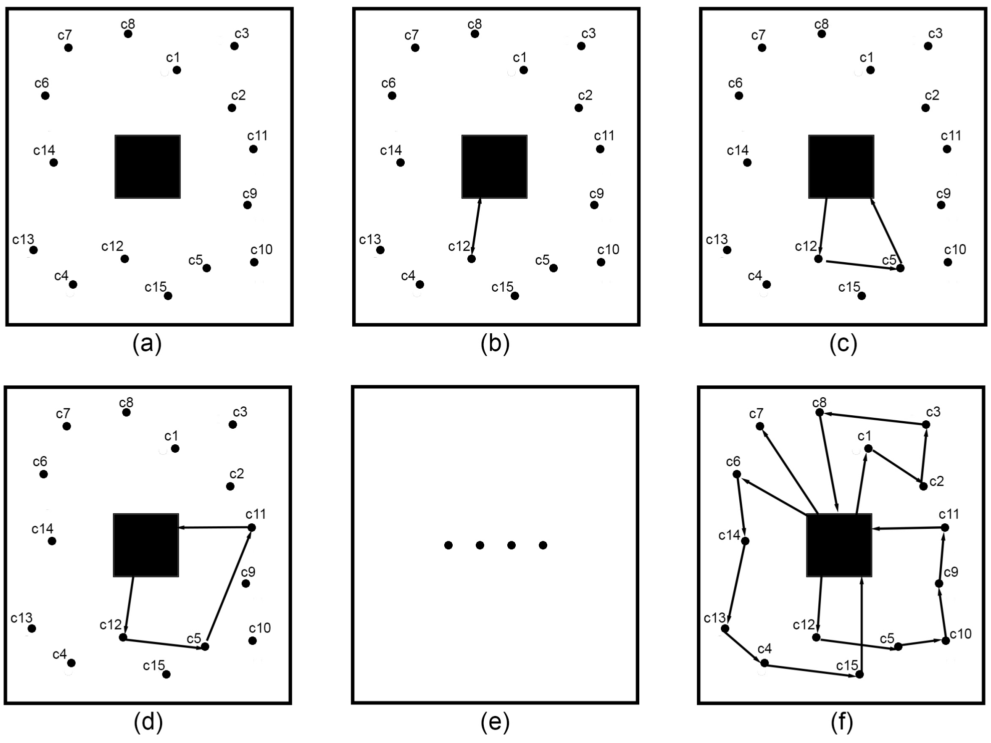

Figure 12: Running the Mole and Jameson Heuristic

feasible customer to be inserted in the open route next to the customer that is most recently visited in the open route. Some complex versions can consider checking the best possible position in the open route by going through the whole route from start to end.

In Laporte and Semet 2001, the Mole and Jameson sequential insertion heuristic is discussed. It starts with an empty route set $Sol = \varnothing$. The current open route is $r_{cur}.S = \varnothing$. The profit of inserting a non allocated customer i in route r_{cur} such that $q_i + demand(r_{cur}) \leq Q$ is calculated. The process is carried out in three steps. First, it is determined what is the minimal detour α_{ip} of the visiting customer in the present open route. For each position $p(1 \leq p \leq |r_{cur}.S| + 1)$ in the current route, i (the cost of inserting) is calculated as $\alpha_{ip} = c_{r_{cur}[p-1]i} + c_{ir_{cur}[p]} - \lambda c_{r_{cur}[p-1]r_{cur}[p]}$ such that λ is a parameter. This α_{ip} is the minimal value that is obtained for some p representing the smallest detour that is possible. Further, the saved distance is evaluated which is obtained by visiting i in the current route instead of its own route. This is achieved by the following equation $\beta_i = \mu c_{0i} + \alpha_{ip}$ such that μ is a parameter. The parameters λ and μ are used to implement several standard insertion rules.

For example, if $\lambda = 1$ and $\mu = 0$, customer yielding minimum extra distance will be inserted. If both of them are zero then the customer corresponding to the smallest sum of the distance between two neighbors will be inserted. Lastly, the minimal detour $r_{cur} = insert(i, r, p)$ is obtained by inserting customer i maximizing β_i at position p of the current route. By using a 3-exchange(3-opt) optimization (Lin 1965), the resulting route is then optimized. Once the current open route gets saturated and can not accommodate more customers feasibly then it becomes part of the current solution. After this, the above-mentioned procedure is applied to a newly selected empty route r_{cur}, and this process is repeated until all customers are allotted a route and the routes are added to the current solution. The number of vehicles is not used as a decision variable in forming the solution so the routes can be more than the vehicles available. Figure 12 exhibits application of the Mole and Jameson Heuristic.

4.5.3 Sweep Heuristic

The sweep heuristic applies to planar instance of the vehicle routing problem. They are also known as clustering algorithms. They are two-phase algorithm. The first phase sorts customers into sets also known as clusters. The second phase then generates one route per cluster, computing the visit to all the customers in that cluster. This step is usually carried out by solving a traveling salesman problem per cluster. The solution may be repaired by employing a third phase if the route formed in the second phase can not be serviced by a single vehicle.

In sweep heuristic, the vertexes in V are considered to be scattered on a plane. Each customer $i \in V \backslash \{0\}$ has specific x-y coordinates w.r.t. the depot (θ_i, ρ_i), while for some customers, $j \in V \backslash \{0\} \theta_j = 0$. It initiates with a given route that is empty $r_{cur}(r_{cur}.S = \varnothing)$. An imaginary ray emanating from the depot is rotated in a circle, passing over the customer position, As soon as the customer v comes across the ray, it gets included in the set comprising of the customers that are visited in the route presently under consideration if $demand(r_{cur}) + q_v \leq Q$. If the customer can not be added to the route then the route is closed and added to the solution, a next route is opened and v is added to this fresh route. After the ray has performed a complete rotation, the present route is added to Sol and the entirety of customers are marked as visited. The optimal sequence for every route in the solution is obtained by solving a

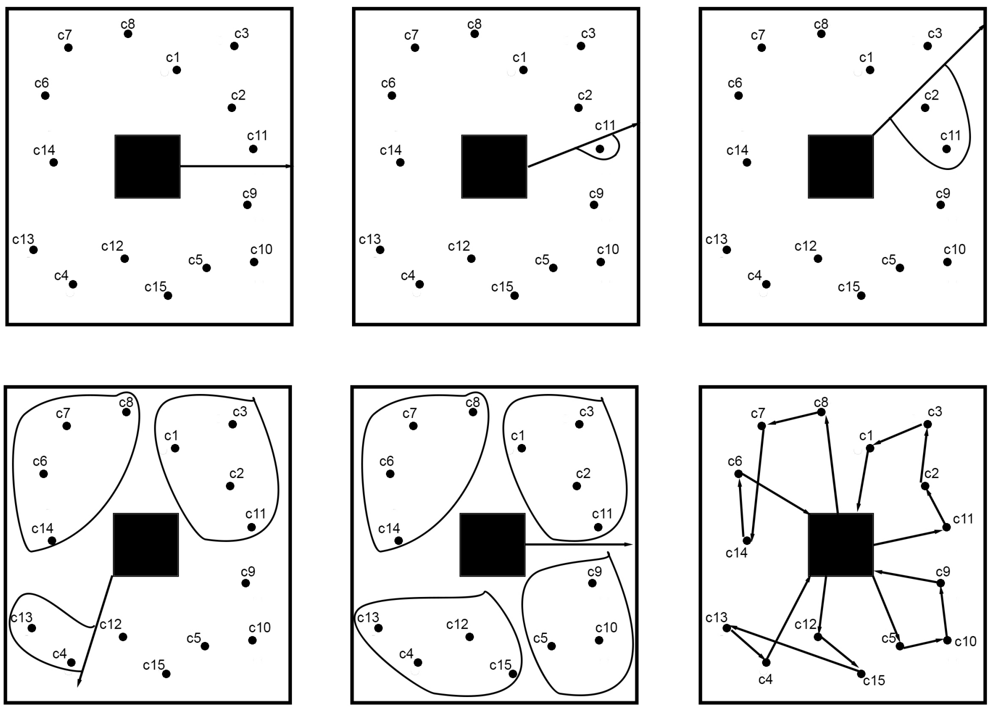

Figure 13: Implementation of the Sweep Heuristic

traveling salesman problem (exactly or approximately). The solution might contain more routes than available vehicles because vehicles are not a decision variable while forming the solution. Figure 13 shows an example of running the Sweep Heuristic on a CVRP.

4.6 Local Search

Local search algorithm tries to solve a problem by moving from solution to solution in the space of candidate solutions. The space contains all possible solutions either feasible or infeasible to the problem and it is also known as solution space. In local search heuristic, we have always got a current solution. It applies local changes to the current solution by a systematic evaluation of the effect of changing the solution. If an improved solution is reached by one of the changes, then the current solution is replaced by the new improved (neighboring) solution. The process is repeated until a solution deemed optimal is found or a processing time-bound is reached. In some variants of local search heuristic, the current solution can be modified even if evaluation signals a negative result. This is done in the hope of finding a much better solution after a few steps in the solution search space. Local search algorithms are useful for solving pure optimization problems, in which we try to find the best state according to an objective function (Artificial Intelligence, chapter 4, by Russell and Norvig 2003b).

Local search can be better explained with the help of the state-space landscape as shown in figure 14 from discussion in Artificial Intelligence, chapter 4, by Russell and Norvig 2003b. The location in the landscape represents the state of the solution and elevation represents the value of heuristic cost function for that state. If elevation is taken as the cost of the solution then the goal is to find the absolute lowest point i.e. global minimum. Otherwise, if elevation corresponds to objective function then the goal is to find the absolute highest point i.e. global maximum. In local search algorithms we search this landscape to find the best possible solution. The state-space may contain locally optimal solutions and a global optimal solution. A local optimum is the optimal solution in the neighborhood surrounding it, meaning that this solution can not be improved by any of the neighboring solutions. Construction and evaluation of the neighboring solution is carried out at each iteration in the local search, one of which is then picked as the new current solution. On the other hand a global optimum is absolutely optimal solution through the whole state-space landscape.

Two concepts related to local search are Intensification and Diversification. In intensification the search in a solution space is focused on a specific portion known to produce a good solution. In diversification the search in a solution space is randomized to explore unknown solution features. This is done to ensure that various areas are explored and there is a high chance of finding the global optimum. It is important to find a good balance between intensification and diversification (Vehicle Routing, chapter 4, by Laporte, Ropke, and Vidal 2014).

Different strategies of local search can be used to prune neighborhoods efficiently. The strategy specifies the next search step to be taken. The selection of strategy to move to a neighboring solution from the current solution is called pivoting rule (Yannakakis 1990). The widely used pivoting rules are the First Improvement and Best Improvement strategies. The first improvement neighbor selection strategy reduces execution time by avoiding to evaluate all neighbors. In this strategy the search examining the neighboring solutions simultaneously

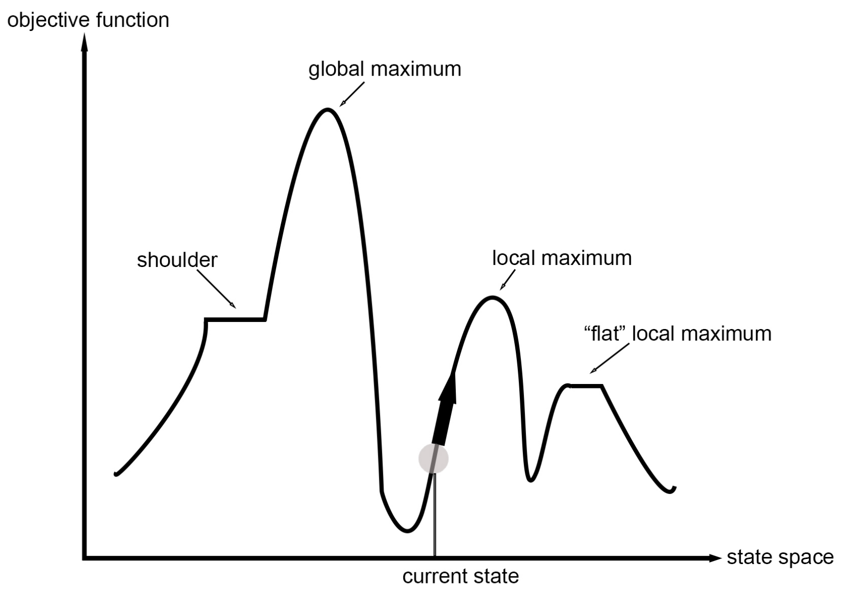

Figure 14: A One-Dimensional State-Space for Local Search

with the construction of the neighborhood and the moment a better neighboring solution is arrived at, it becomes the current solution. The order in which neighbor solutions are evaluated can have a vital impact on the efficiency of this search strategy. We can use fixed ordering or random ordering for evaluating the neighboring solutions. For fixed evaluation orderings, repeated runs starting from the same initial position will give the same local optimum as a result. Using random evaluation ordering many different local optimums can be reached hence diversifying the search process and producing a chance of obtaining the global optimum. In greedy hill-climbing or discrete gradient descent the complete neighborhood is constructed and evaluated. In each iteration, a solution that is improving the quality the most is opted to be the new current solution.

4.6.1 Adaptation to the CVRP

Local Search is vastly used in vehicle routing problems and has provided promising solutions particularly in examples of large magnitude in which a feasible solution could not be yielded by following exact methods. The remainder of the section illustrates the adaption of the main steps of the local search for CVRP. The discussion in this section is based on Laporte and Semet 2001, Funke, Grünert, and Irnich 2005, Thomas, Smet, and Deville 2015 and Massen 2013. The idea of notations is inspired by Massen 2013.

1. **Making of the initial solution**

   Construction heuristics are used in building the initial solution as exhibited in the section 4.5. For the purpose of generating a number of initial solutions for a given set of customers, we can take some meaningful random steps. An example of a random step could be to penalize an arc in the current solution if the arc was present in earlier formed solutions. It is important to decide at this stage that the initial solution should be forced to be feasible according to our requirements or not. In the scenario of a feasible solution, the number of routes should be similar to the number of vehicles in the fleet, or every route should satisfy the capacity constraints.
2. **Solution evaluation**

   In case of a feasible solution, generally evaluation of the solution's objective function is done. In local search, where infeasible solutions are also stored to the graph, a modified objective function is used for evaluation. An infeasible solution may for instance build more routes than the number of vehicles in the fleet, or some route might cross the capacity threshold of the vehicle providing service to the route. The modified objective function also captures the in-feasibility, as for example crossing the capacity threshold of the vehicle in a route. The purpose of storing an infeasible solution is to make the solution feasible by restructuring the solution, later on, if no feasible solution was found directly. The search for feasible solutions can be made more effective by penalizing infeasible solutions.
3. **Construction of the neighborhood**

   There are two major categories of VRP neighborhoods: Single-Route Improvements and Multi-Route Improvements, as mentioned in Laporte and Semet 2001. As evident from the name in single-route improvements changes are made to one route at a time. We make change by moving customers in the route sequence. In multi-route improvements, customers are exchanged and moved between two or more routes at a time. The impact of multi-route improvements on the structure of the CVRP solution is greater as compared to Single-Route Improvement.

Some of the major decisions taken while constructing the neighborhood are consideration of infeasible solutions, computation of reduced neighborhood or full neighborhood, and the usage of some neighborhood operator. If it is allowed to employ an infeasible solution as the initial solution then the search procedure should be allowed to move to an infeasible neighboring solution. This decision is taken because in the proximity of an initial infeasible solution, it is unlikely to find a feasible solution. Another possibility is to restrict the search from moving to an infeasible solution once we have found a feasible solution. After finding a feasible solution we work towards improving the quality of the solution.

The neighborhood is constructed with the help of operators. Each operator applies a specific move to the current solution to obtain a neighboring solution. The efficiency of local search is highly dependent on the neighborhood operators used, which highlights the point that they should be selected with great care and consideration. Some of the vital neighborhood operators used in local search procedures for CVRP are given below.

(a) ***Relocate operator***

In relocation, one node is moved from its current route and inserted in a different route. To relocate a customer i currently part of $r_1 (i \in r_1.S)$ by removing it from r_1 and placing it in route r_2 at the position p such that $r_2(1 \leq p \leq |r_2.S|+1)$, we use the relocate operator $\eta_{reloc}(i, r_1, p, r_2, Sol)$ $(r_1, r_2 \in Sol)$.

(b) ***Swap operator***

In swap move, one node is moved from the first route to the second route and a second node from the second route to the first route. It can be seen as double relocation in which customers are inserted at the counterpart's position. To swap i and j (customers) that are part of unique routes r_1 and r_2 $(r_1 \neq r_2, i \in r_1.S\,, j \in r_2.S)$ and exchange them, we use the swap operator $\eta_{swap}(i, r_1, j, r_2, Sol)$ $(r_1, r_2 \in Sol)$

(c) ***k-exchange operators***

The K-exchange operator drops k edges in the same route and then reconnecting the resulting segments by other edges. The purpose of this operator is to reconnect the segments in a different order. For the division of a route r_1 into segments $(k+1)$ by deleting all arcs in A_1 from $r_1(A_1 \subseteq \bigcup_{j=0}^{|r.S|}\{(r[j], r[j+1])\})$ and rejoining the resulting segments using the arcs in A_2 $(A_2 \subseteq \bigcup_{j=0}^{|r.S+1|}\{(\delta^{-}_{r[j]} \cap \bigcup_{l=0}^{|r.S|+1}\delta^{+}_{r[l]})\})$, we use the k-exchange operator $\eta_{kex}(r_1, A_1, A_2, Sol)$ $(r_1 \in Sol\ and\ A1 \cap A2 = \varnothing)$.

(d) ***Cross operator***

The cross operator cuts two different routes in two parts and recomposes them by crossing edges. For exchanging the section, the starting points of which are i and j and ending is the depot, for route r_1 and route r_2 $(r_1 \neq r_2, i \in r_1.S, j \in r_2.S)$, we use the cross operator $\eta_{cross}(i, r_1, j, r_2, Sol)$ $(r_1, r_2 \in Sol)$.

4.7 Metaheuristics

Metaheuristic is a higher level heuristic used to select another heuristic that may provide a better solution to an optimization problem. Metaheuristic is a master strategy guiding and modifying subordinate heuristics by an intelligent combination of various concepts for exploring the search space. It is useful especially when we have very little or incomplete information in advance or limited computational power to know the global optimum solution to a problem (Essentials of Metaheuristics, chapter 1, by Luke 2013). They are not problem-specific algorithms, still, its intrinsic parameters need some fine-tuning for adapting the technique to the given problem. They are usually non-deterministic in nature. Metaheuristic is about optimizing the solution of a problem towards a better state by avoiding getting stuck in local optima but it does not ensure finding the globally optimum solution. The advantage of metaheuristics is the use of less computational power as compared to other optimization problems.

In Vehicle Routing, chapter 4, by Laporte, Ropke, and Vidal 2014, metaheuristic is broadly categorized into local search methods and population-based methods. Local Search Methods explore the neighborhood of a solution in subsequent iterations. Some of the examples of methods employing local search are simulated annealing, tabu search, iterated local search and variable neighborhood search. Population-based heuristics are used to generate better solutions by evolving and combining a population of solutions, which works on the principle of survival of the fittest. Among other examples, ant colony optimization is an instance of population-based heuristics.

4.8 Ant Colony Optimization

The discussion of Ant Colony Optimization (ACO) is based on Dorigo, Maniezzo, and Colorni 1996, Dorigo, Birattari, and Stutzle 2006, Vehicle Routing, chapter 4, by Laporte, Ropke, and Vidal 2014 and Massen 2013. The idea of notations is based on Dorigo, Birattari, and Stutzle 2006. The food searching behavior of ant colonies has been intensively studied and finds various applications in solving other real-world problems. Ants communicate with each other about the path leading to food or their colony with the help of pheromones. They produce and emit these hormonal chemicals for relaying a message to another member of the colony. They produce different types of pheromones, each with its own purpose. The pheromones are emitted to attract mates, to signal danger to the colony and to give path directions about a location. The pheromones that give path directions about a location are also known as trail pheromones. This pheromone is emitted by ants when they return to the colony with food to attract other ants for following the path to food location. When making a choice between various paths, ants prefer to choose the path with the highest pheromone presence. Between the colony and location of food, the level of pheromone will rise rapidly on the path that is shorter for the reason that it will take lesser time to cover it. As a result, the ants opt and go on the shorter path, in which process even more pheromones are deposited; consequently, the majority of ants follow the shorter path.

In the above example, ants behave as intelligent agents in carrying out the optimization of the problem by constructing solutions. Ants start building a solution from scratch and extend it in several steps towards a full solution. At each step there is a probability for selecting from different path options to further enhance the current solution into a new partial solution. The way pheromones are deposited by ants and afterward, its evaporation provides the decision-making process to modify the current solution forming method. Each ant continues to work until it reaches the full solution or hits a dead-end in enhancing the current solution in a feasible way at any point. Local search is usually used to optimize the solutions. In each iteration of ant colony optimization, a whole colony consisting of m ants is executed. Pheromones are deposited on the paths the ants used and as a result the solution is possibly updated. This is applied in proportional measure to the quality of the given solutions. ACO can be used to solve vehicle routing problems due to the similarity of finding an optimized set of paths in the problem graph. A number of CVRP variants have been solved by the ACO approach as discussed in Reimann, Stummer, and Doerner 2002, Zhang et al. 2006 and Fuellerer et al. 2010.

4.8.1 Application of the Ant Colony Optimization in CVRP

We discuss basic ant colony optimization system where the ants use a simple insertion heuristic. Following are the design choices that need to be considered in applying ACO for finding a solution to a particular problem:

1. **Constructing the solution**
   To construct the solution each ant executes the construction heuristic as discussed in section 4.5. The depot is the starting point of an ant and the next vertex is selected by the ant along each step. Hence only one route is traversed by an ant at a time. The ant has to decide to choose a destination vertex from unchosen vertexes or move back to the depot at each step. If at any step the ant chooses to move back to the depot, the current route is closed and in the next step a new route is started. With the construction heuristic the ant can build more routes than the available vehicles. If the routes are more in number than the available vehicles, we can either consider it an infeasible solution or post optimize this infeasible solution using local search to make it feasible.
2. **Selecting the next step**
   The ant chooses a new destination vertex at each step, which is analogous to choosing and adding an arc to its path. The choice of the next arc is random but still tending towards arcs with greater pheromone quantity. Let i be the last vertex that has been added by the ant to its route and s^p the solution with the present set of closed routes. For the feasible extension of the current route the set of vertices is given as $\mathbf{N}(s^p)$, which is then defined as edges (i,l) where l is a vertex not yet visited by the ant k. The

following equation gives associated probability for visit of vertex j as the next vertex

$$p_{ij}^{k} = \begin{cases} \frac{\tau_{ij}^{\alpha}.\eta_{ij}^{\beta}}{\sum_{c_{il} \in \mathbf{N}(s^{p})} \tau_{il}^{\alpha}.\eta_{il}^{\beta}} & if c_{ij} \in N(s^{p}), \\ 0 & otherwise, \end{cases}$$

The parameter α and β control the relative importance of the pheromone in comparison to the heuristic information η_{il}, which is given by:

$$\eta_{ij} = \frac{1}{d_{ij}},$$

where d_{ij} is the distance between vertex i and j.

3. **Post-optimizing the solution**

   Local search is commonly used for the post-optimization of the solution that the ants have built. (as seen in section 4.6).

4. **Updating the pheromone deposit**

   After finishing one complete iteration of the ant colony optimization, the pheromone deposit on the graph is updated. The formula below exhibits the updating of deposit of pheromone τ_{ij} on arc (i, j)

$$\tau_{ij} \leftarrow (1-\rho).\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k},$$

   where ρ is the evaporation rate, m is the number of ants, $\Delta\tau_{ij}^{k}$ is the quantity of pheromone laid on arc (i, j) by ant k:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & if\ ant\ k\ used\ edge\ (i,\ j)\ in\ its\ tour, \\ 0 & otherwise \end{cases}$$

   Where Q is a constant, and L_k is the length of the solution tour developed by ant k

5 Combining Vehicle Routing And Bin Packing

In this chapter, we combine the vehicle routing and three-dimensional bin packing problem. In order to do that, the three-dimensional capacitated vehicle routing problem is introduced first. The experimental setup is described for combining vehicle routing and bin packing. Finally, the results after solving the problem are discussed.

5.1 Introduction

The packing and delivery of goods to different customer locations are two important transportation logistics operations. Its effective execution saves the operational cost of the distribution and manufacturing companies plus also satisfies the needs of a vast variety of customers to whom the goods need to be delivered. Packing and transportation of goods can be quite interdependent for companies working in the field of logistics distribution. For example, there is little advantage in making an optimal route from a combination of different customer locations while the demands of the respective customers can not be fully loaded in a vehicle assigned to the route. It would be uneconomical to use multiple vehicles for a route while we can use a single one, if we pack the goods in an efficient way.

5.2 Combining Vehicle Routing and Bin Packing

The consideration of capacity limits of vehicles in a vehicle routing problem is generally called the Capacitated Vehicle Routing Problem (CVRP). It is about delivering service to a given set of customers with associated demands employing a given fleet of vehicles. Every vehicle is considered to have limited capacity. In the basic version, all vehicles start and end their journey at the depot. The goal is to divide the customers into as few routes as possible hence minimizing the distance. It should be kept in focus that the truck capacity for a route does not exceed the threshold limit of the truck. The majority of vehicle routing problems consider the capacity constraint as a threshold of weight or volume carried by a vehicle.

Two-dimensional Capacitated Vehicle Routing Problem (2LCVRP) generalizes the CVRP, in which an item is only represented by one positive integer, representing weight or volume. In 2LCVRP, the shape of items and vehicles is considered in two dimensions i.e. width and height. The 2LCVRP plays an important role in real scenarios of logistics in which the product's shape is considered. 2LCVRP is useful in real-life scenarios like transporting large kitchen appliances such as refrigerators or catering equipment like food trolleys (Iori and Martello 2010).

Vehicle routing combined with three-dimensional (width, height, depth) bin packing problem is known as Capacitated Vehicle Routing Problem with three-dimensional Loading Constraints (3LCVRP). This problem was first discussed in Gendreau et al. 2006, where 3D rectangular loading space replaces the scalar capacity of the vehicles. The goal is to reduce the traveling cost and also to provide each vehicle with a feasible loading plan. Each item in 3LCVRP is considered as a cuboid with weight. In general, a cuboid is an object with dimensions as width, height and depth. It has six rectangular faces. The vehicle volume capacity is expressed as a 3D loading space. The 3LCVRP is a challenging optimization problem because both vehicle routing and 3D bin packing are hard to solve practically. Gendreau et al. 2006 states the 3LCVRP problem to be NP-hard.

In this thesis the focus is on solving a variant of 3LCVRP, which is a Heterogeneous Fleet of Pick Up and Delivery Problem with Time Windows and three-dimensional Loading Constraints (3L-HFCVRPTW). In 3L-HFCVRPTW the solution ensures that item can be picked up and delivered to clients in a specific time range while the packing is feasible for the 3D loading space of the vehicle. All the vehicles are not identical in size. The pick up of an item should always come first than its delivery. This problem is defined in section 5.4.

5.3 Related Work

The three-dimensional capacitated vehicle routing problem, was first addressed by Gendreau et al. 2006. They modeled "sequence-based loading", "stacking" and "vertical stability constraints". A set of test instances was introduced, which can be used by other researchers for bench-marking the performance of different heuristic and meta-heuristic solutions. The test instances are available on the link `http://or.dei.unibo.it/`. There are 27 test instances with varying number of customers, vehicles and items. The maximum number of customers, items and vehicles is 100, 198 and 26 respectively. The graphs, customers demand and vehicle weight capacity are taken from 27 Euclidean CVRP instances (see The Vehicle Routing Problem, chapter 1, by Toth and Vigo 2001 for a detailed description of CVRP testbed instances).

This paragraph discusses other research articles that solved the problem discussed in Gendreau et al. 2006 and tested their approach on the testbed provided with it along with some new test instances. The discussion is based on a survey by Pollaris et al. 2015. In Fuellerer et al. 2010, the problem was solved using Ant Colony Optimization. The vehicle routing and bin packing problem is solved by combining two different heuristic information measures. This showed an improvement of 6.43% over Tabu Search in Gendreau et al. 2006. In Tarantilis, Zachariadis, and Kiranoudis 2009, a hybrid of Guided Local Search (GLS) and Tabu Search (TS) is used to solve the three-dimensional capacitated vehicle routing problem. This showed an improvement of 3.13% over Tabu Search in Gendreau et al. 2006. In Ruan et al. 2013, Honey Bee Optimization is used to solve the problem. It improved the quality of solution on average 7% over Tabu Search in Gendreau et al. 2006 and improved in certain instances from previous results of GLS in Tarantilis, Zachariadis, and Kiranoudis 2009 and ACO in Fuellerer et al. 2010. In Miao et al. 2012, a hybrid approach is introduced, which combines Genetic Algorithm for vehicle routing problem and Tabu Search for three dimensional loading. This hybrid genetic algorithm obtains new best solutions for several instances. Ren, Tian, and Sawaragi 2011 proposed the application of a Branch and Bound method to the modified 3L-CVRP, which consisted of relaxing and replacing the loading constraints by a volume-ratio constraint. In the next step, a container loading algorithm, essentially a tree search method is used to check whether the items for the generated routes can be loaded into the vehicles. This process is repeated and volume-ratio is varied until all items are feasibly loaded. It also improved previous best results for certain cases in the testbed.

In this paragraph we discuss research articles published on Vehicle Routing Problem involving Pickup and Delivery (VRPPD) with multiple vehicles and loading constraints. The discussion is based on a survey by Pollaris et al. 2015. Cherkesly, Desaulniers, and Laporte 2015 proposes an exact solution. It presents a formulation for VRPPD with time windows and last in first out (LIFO) constraints. For cases with a maximum of 75 requests, three Branch-Price-and-Cut algorithms are developed to find an exact solution. Fagerholt et al. 2013 formulated VRPPD with “time windows”, “complete-shipment constraints” and “connectivity constraints” to solve problems of tramp shipping. Tramp shipping refers to the trade practice of ships and boats, not having a fixed schedule or published ports. They trade on the spot market with no fixed schedule. A Tabu Search (TS) heuristic is proposed to solve the problem. In Cheang et al. 2012, a solution is proposed to Multiple Vehicle Pickup and Delivery problem with LIFO loading and distance constraints (MTSPPD-LD). It devises a two-stage approach for solving the problem. In the first phase, using Simulated Annealing and Ejection Pool, the number of vehicles is decreased. In the second phase using Variable Neighborhood Search and Probabilistic Tabu Search, total travel distance is decreased. Malapert et al. 2008 suggests a solution to two dimensional loading and VRPPD with multiple vehicles and sequence-based loading. It develops a constraint programming loading model based on a scheduling approach. The article highlights the incompatibility of reduction procedures with sequence-based loading. In Männel and Bortfeldt 2016, a solution is proposed for VRPPD with three-dimensional loading problem and homogeneous fleet. The routing procedure alters a renowned Large Neighborhood Search used for the 1D-PDP (Ropke and Pisinger 2006). A tree search heuristic is employed for loading the boxes in vehicles.

5.4 Problem Definition

The problem is defined on a graph $G = (V, E)$ representing the road network. Let E be a set of undirected edges (i, j) that connects all node pairs $(0 \leq i\,,\, j \leq 2n\,,\, i \neq j)$. The set of vertexes $V = \{0, 1, \ldots, n, n+1, \ldots, 2n\}$ represents all the vertexes, i.e. the points for pickup and delivery as well as the depot locations at 0. For each edge $e_{ij} \in E$, there is an associated routing cost c_{ij} between v_i and v_j and the cost of edges is symmetric i.e. $c_{ij} = c_{ji}$ $(0 \leq i\,,\, j \leq 2n\,,\, i \neq j)$. We have got n customer requests and each request is identified as i consisting of a pickup point i, a delivery point $n+i$, and a set I_i of goods that are to be transported from the pickup point to the delivery point. The pickup point should always come first before the delivery point for the same customer request. A set of goods I_i includes m_i cuboid pieces and each I_{ik} has length l_{ik} , the width w_{ik} and the height h_{ik} $(i = 1,\dots,n, k = 1,\dots,m_i)$. The total weight of m_i items is d_i. Every customer has a time window $[a_i,b_i]$ where a_i is the beginning and b_i is the end of the time window. We assume that $a_i \geq 0$ and $b_i > 0$. $[a_0,b_0]$ is the time window of the depot, and each vehicle starts the journey from the depot at instant $a_0 = 0$ which shows the beginning of the depot time window and must come back to the depot before instant $b_0 > 0$, which is the end of the depot time window. A heterogeneous fleet of vehicles is located at the depot. Each different vehicle is depicted by $t(t = 1,\dots,T)$, where T depicts the total number of different types of vehicles. Each vehicle has a weight capacity D_t, fixed cost F_t, unit travel cost V_t and 3D rectangular loading space of length L_t, width W_t and height H_t. Each vehicle has an opening at the rear door that is as large as the W_t x H_t plane.

A valid route r is a sequence of $2p+2$ nodes $(p \geq 1)$ and contain a minimum of four nodes with start and endpoints at a depot. This route r includes the pickup and delivery points of p different request among the total requests n. There might be a single hub in the route to which items are delivered. A customer pickup and delivery request should only appear once in the route sequence. By adding the cost of all edges that are part of a route we get the total cost of the route.

In 3L-HFCVRPTW, a valid solution ensures that items can be picked up and delivered to clients through a set of feasible routes in a specific time range while the packing is feasible for the 3D loading space of the vehicle. Solution cost is calculated by total distance and driving time costs, ferry costs, penalties for being late, hub handling costs and penalties if vehicles are driving outside their dedicated areas. We are visiting hubs because we can change the package from one vehicle to another.

5.5 The Hybrid Algorithm

For the 3L-HFCVRPTW, we experiment with a hybrid algorithm comprising of two separate procedures for routing and packing. The procedure for routing is developed by professor Prof. Olli Bräysy of Weoptit Oy. Boxes are placed into vehicles by using a slight variation of Largest Area Fit First (Gürbüz et al. 2009).

5.5.1 Routing Algorithm

The routing phase of the 3L-HFCVRPTW is dealt with by a variant of ejection chain based route reduction for solution construction and ejection based local search for solution improvement. Ejection chains are introduced in Glover 1992 and Glover 1996. The general idea of the ejection chain is applying changes to the state of a set of selected elements, which in this case is the movement of vertices between different routes. This results in identifying collections of other sets, in which the element of at least one set must be “ejected from” their current states. This can trigger a cascading sequence of events analogous to the domino effect. A constructive heuristic starts solving a problem from scratch (see section 4.5). In the process of constructing routes the goal is to keep the distance minimum in the solution. Route construction heuristic work towards the solution of the problem by inserting one customer at a time into partial routes till the construction of a feasible solution. Routes that do not contain any customer are deleted from the solution. This heuristic continues merging routes until no more merging is possible. Subsequent to the creation of the initial solution, the optimization phase starts executing until a time limit is reached or if no further improvement is possible. We are checking for every feasible route that the packing is also feasible.

Any move of customer node incurring penalties is always accepted if it is giving us a better solution. If no moves without penalties are found the first move improving the objective function is accepted or the best move found when there is no other move improving the solution. The chosen move is executed and recorded in a list which contains all the previously executed moves.

Following are the kind of moves considered (r_0 and r_1 being two distinct routes):

1. **Shifting the customer nodes:**
   Insert pickup and delivery point of customer $i \in r_0$ at a specific position in r_1 and remove it from r_0.
2. **Swapping the customers nodes in different routes:**
   Given $i \in r_0$ and $j \in r_1$, swap i with j or it can move customers in one swap. A small segment of pickups and corresponding deliveries or small segment of deliveries and their corresponding pickups.

3. **Swapping customers nodes within a route:**
   Swap two customers pickup points or their corresponding deliveries in a route i.e. customers $i, j \in r_0$.

The order in which the moves happen is as follows. Firstly all the possible shift moves are evaluated for different routes. Secondly all the possible insertion points are tried in order to evaluate all swap moves. The iteration of the routes is carried out in random order for any particular type of move. Swapping of customer nodes within a route is done through a selective approach by carrying out a random evaluation of swap moves. If at any point, there is a feasible neighbor in a feasible mode, we skip the rest of the moves not leading to a better solution. Construction heuristics gradually builds a feasible solution while tracking the current cost of the solution but they lack an improvement phase by itself (Laporte and Semet 2001). For improvement, an ejection based local search is employed to improve the solution as much as possible.

5.5.2 Packing Algorithm

A valid plan of packing for 3D loading space of vehicles consists of one or multiple placements and the following conditions hold: Each box lies completely within the 3D loading space of the vehicle; Any two boxes placed in a vehicle do not overlap with each other; the total weight of a packing plan should not cross the vehicle limits.

The 3D packing algorithm used is a slight variation of the Largest Area Fit First (LAFF) minimizing height (Gürbüz et al. 2009). It uses a heuristics that carries the placement of the boxes such that those with the largest surface area are placed first while keeping the height minimum from the floor of the container. If the shape of an item is not cuboid then the bounding box of that item is considered e.g. cylinders. A small volume penalty is introduced for items required to be tied with ropes. The width, depth and height of the container is fixed as per the specification of the vehicle available. A vehicle can have two containers for packing items i.e. a hauler and trailer.

When the height, width and depth values of the container are determined, the specified boxes are packed using two placement techniques. The first placement method places a box in the container to start a new packing layer and set a specific height for the layer. In this approach, those boxes that have the largest surface area are sorted in descending order. The chosen boxes are examined to find a box having the minimum height. Then, this box is packed in the container while keeping its largest surface parallel to the bottom of the container. The space for the remaining boxes is allocated by the second placement method ensuring that the boxes are placed in the same layer while not crossing the height of the layer. In this method the boxes with the largest volume are placed first in the empty spaces around the box placed in the first placement method. When no more boxes can be placed in the current layer using the second placement method, a new layer is started using the first placement method.

If we find a feasible packing solution, we proceed to the route optimization stage for the selected customers. Otherwise we remake our selection of customers for a new route.

5.6 Experiments

5.6.1 Experimental Setup

The program is coded in Java. The vehicle routing code[1] is refactored, and combined with a slightly modified version of bin packing code[2]. The coded program is run on a computer with the following configuration: Intel Core i5 2.30 GHz 8 GB Ram.

1. **First Dataset:**
   The first dataset was created from a file with a set of Pick up and Delivery (PDP) customers that was provided for testing and integration by the company (Weoptit Oy) accompanied by other files for vehicles, ferries, terminals and distance matrix. The dimensions of width, height and length of items in the PDP file were erroneous for some customers i.e. they were bigger than the respective dimension of vehicles. There were other errors such as the volume of the object was not matching the product of width, height and length. To remove this error, the volume was regenerated from dimensions present. Another group of items had volume given but the dimensions were not known, in this case, the dimensions were inferred from the volume by taking a cube root and

1. Developed at Weoptit Oy. I have refactored it to work with 3D bin packing.
2. https://github.com/skjolber/3d-bin-container-packing

then clipping the respective dimension as per the minimum value of that dimension for containers and adding it to another dimension of the object. In this process, if the volume can not be broken down into length, width or height without surpassing a minimum capacity of that dimension in the vehicles present, then that dimension was manually corrected and the volume regenerated from the new dimensions. The minimum length, width, height in vehicles present was 13.6 unit, 2.45 unit, 2.65 unit respectively. The two types of corrections that were applied to items when the volume was given but the dimensions were missing are as following:

   (a) **Type 1 corrections:**
   The cube root of the volume was calculated for items whose length, width and height dimensions are missing, and the upper limit of 2.4 unit was maintained on the width and 2.6 unit was maintained on the height. The clipped value is adjusted in length = volume / (width * height).

   (b) **Type 2 corrections:**
   The cube root of the volume was calculated for items whose length, width and height dimensions are missing, and the upper limit of 2.3 unit was maintained on the width and 2.4 unit was maintained on the height. The clipped value is adjusted in length = volume / (width * height)

2. **Second Dataset:**
   The Second dataset was created from a file with a set of PDP customers data that was provided for testing and integration by the company accompanied by other files for vehicles, ferries, terminals and distance matrix. In this the dimensions are randomly generated on the basis minimum dimensions of vehicles present, which was 13.6 unit, 2.45 unit, 2.65 unit of length, width, height respectively. The length, width and height for customer objects was generated in the ranges $[0.40, 13.60]$, $[0.40, 2.45]$, $[0.40, 2.65]$ respectively. The volume is calculated as the product of the three dimensions.

5.6.2 Results for Computational Time

The processing time (seconds) consumption of integrating the 3D bin packing with vehicle routing pickup and delivery along with the corresponding time slots are given in tables 2 and 3. The same experiment was run three times. The reported time intervals are obtained by taking an average over three runs and then rounding to the nearest number. The general trend can be easily seen in the results that the packing algorithm integrated with the routing algorithm does not incur a huge time consumption penalty. There are a small number of anomalies in this trend due to the nature of used heuristics.

File	***Customers***	***Vehicles***	***Running Time Without Packing***	***Running Time With Packing***
Type1	138	11	28	53
Type2	138	11	24	26

Table 2: Computational time results based on first dataset

File	***Customers***	***Vehicles***	***Running Time Without Packing***	***Running Time With Packing***
Random1	138	11	22	62
Random2	138	11	27	42
Random3	138	11	22	33
Random4	138	11	19	27
Random5	138	11	22	49
Random6	138	11	20	29
Random7	138	11	17	35
Random8	138	11	20	19
Random9	138	11	22	29
Random10	138	11	19	16

Table 3: Computational time results based on second dataset

5.6.3 Results for Vehicle Routing Optimization

The routing solutions of integrating the 3D bin packing with vehicle routing pickup and delivery along with their corresponding virtual vehicles and unassigned order values can be seen in tables 4 and 5. The same experiment was run three times. The reported virtual vehicles and unassigned orders are obtained by taking an average over three runs and then rounding to the nearest number. In our solution, the concept of virtual vehicle is used. An order is assigned to a virtual vehicle if it cant to be accommodated in the real vehicles available for the vehicle routing solution. An order assigned to a virtual vehicle is in essence an unassigned order in the real world. The general trend can be seen that the introduction of approximate 3D bin packing increases the number of virtual vehicles and unassigned order in most of the cases. There are a small number of anomalies in this trend due to the nature of used heuristics. This shows that approximate 3D bin packing gives us a more realistic solution that can be implemented on the ground, saving our time, resources and manual labor.

File	***Customers***	***Vehicles***	***Virtual Vehicles With Packing***	***Virtual Vehicles Without Packing***	***Orders Unassigned With Packing***	***Orders Unassigned Without Packing***
Type1	138	11	53	40	88	62
Type2	138	11	60	40	98	62

Table 4: Vehicle routing optimization results based on first dataset

File	*Customers*	*Vehicles*	*Virtual Vehicles With Packing*	*Virtual Vehicles Without Packing*	*Orders Unas-signed With Packing*	*Orders Unas-signed Without Packing*
Random1	138	11	48	41	62	70
Random2	138	11	42	37	68	60
Random3	138	11	51	34	82	58
Random4	138	11	49	43	81	76
Random5	138	11	41	47	65	72
Random6	138	11	43	38	69	62
Random7	138	11	48	46	74	66
Random8	138	11	59	50	82	80
Random9	138	11	51	43	78	68
Random10	138	11	51	43	83	66

Table 5: Vehicle routing optimization results based on second dataset

6 Conclusion

We have integrated a 3D bin packing algorithm which is a slight variant of Largest Area Fit First (LAFF) minimizing height with a variant of 3LCVRP, which is a Heterogeneous Fleet of Pick Up and Delivery Problem with Time Windows and three-dimensional Loading Constraints (3L-HFCVRPTW). The focus was to introduce a close to real-world 3D packing solution for vehicles fulfilling orders on the routes. In our case, the high speed of the 3D bin packing algorithm is vital because the vehicle routing optimization itself is a processing heavy task especially when the number of customers increases. We have chosen speed over accuracy as we are using rectangular bounding boxes for any item to be packed. The result of this is a more feasible filling of vehicles under human supervision.

Bibliography

Agarwal, Yogesh, Kamlesh Mathur, and Harvey M. Salkin. 1989. “A set-partitioning-based exact algorithm for the vehicle routing problem”. *Networks* 19 (7): 731–749. ISSN: 1097-0037. doi:10.1002/net.3230190702. http://dx.doi.org/10.1002/net.3230190702.

Aggarwal, Divya, and Vijay Kumar. 2019. “Mixed integer programming for vehicle routing problem with time windows”. *International Journal of Intelligent Systems Technologies and Applications* 18 (1-2): 4–19.

Archetti, Claudia, M Grazia Speranza, and Daniele Vigo. 2014. “Vehicle routing problems with profits”. *Vehicle Routing: Problems, Methods, and Applications* 18:273.

Astolfi, A. 2006. *Optimization An introduction.* 1st edition. Chapter 1.

Bräysy, Olli, and Michel Gendreau. 2005. “Vehicle routing problem with time windows, Part I: Route construction and local search algorithms”. *Transportation science* 39 (1): 104–118.

Bula, Gustavo Alfredo, Fabio Augusto Gonzalez, Caroline Prodhon, H Murat Afsar, and Nubia Milena Velasco. 2016. “Mixed integer linear programming model for vehicle routing problem for hazardous materials transportation”. *IFAC-PapersOnLine* 49 (12): 538–543.

Caprara, Alberto, and Michele Monaci. 2009. “Bidimensional packing by bilinear programming”. *Mathematical programming* 118 (1): 75–108.

Çetinkaya, Cihan, Ismail Karaoglan, and Hadi Gökçen. 2013. “Two-stage vehicle routing problem with arc time windows: A mixed integer programming formulation and a heuristic approach”. *European Journal of Operational Research* 230 (3): 539–550.

Cheang, Brenda, Xiang Gao, Andrew Lim, Hu Qin, and Wenbin Zhu. 2012. “Multiple pickup and delivery traveling salesman problem with last-in-first-out loading and distance constraints”. *European journal of operational research* 223 (1): 60–75.

Chen, CS, Shen-Ming Lee, and QS Shen. 1995. “An analytical model for the container loading problem”. *European Journal of operational research* 80 (1): 68–76.

Cherkesly, Marilène, Guy Desaulniers, and Gilbert Laporte. 2015. "Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and last-in-first-out loading". *Transportation Science* 49 (4): 752–766.

Clarke, G., and J. W. Wright. 1964. "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points". *Oper. Res.* (Institute for Operations Research)(the Management Sciences (INFORMS), Linthicum, Maryland, USA) 12, number 4 (): 568–581. ISSN: 0030-364X. doi:`10.1287/opre.12.4.568`. `http://dx.doi.org/10.1287/opre.12.4.568`.

Coffman Jr, Edward G, Michael R Garey, and David S Johnson. 1984. "Approximation algorithms for bin-packing—an updated survey". In *Algorithm design for computer system design,* 49–106. Springer.

Coffman, Edward G, Jr, Michael R Garey, David S Johnson, and Robert Endre Tarjan. 1980. "Performance bounds for level-oriented two-dimensional packing algorithms". *SIAM Journal on Computing* 9 (4): 808–826.

Cordeau, J.-F., G. Desaulniers, J. Desrosiers, M. M. Solomon, and F. Soumis. 2001. "The Vehicle Routing Problem". Chapter VRP with Time Windows, edited by Paolo Toth and Daniele Vigo, 157–193. Philadelphia, PA, USA: Society for Industrial / Applied Mathematics. ISBN: 0-89871-498-2. `http://dl.acm.org/citation.cfm?id=505847.505854`.

Dantzig, G. B., and J. H. Ramser. 1959. "The Truck Dispatching Problem". *Management Science* 6 (1): 80–91. doi:`10.1287/mnsc.6.1.80`. eprint: `http://dx.doi.org/10.1287/mnsc.6.1.80`. `http://dx.doi.org/10.1287/mnsc.6.1.80`.

Darapuneni, Yoga Jaideep. 2012. "A Survey of Classical and Recent Results in Bin Packing Problem".

Den Boef, Edgar, Jan Korst, Silvano Martello, David Pisinger, and Daniele Vigo. 2005. "Erratum to "The three-dimensional bin packing problem": Robot-packable and orthogonal variants of packing problems". *Operations Research* 53 (4): 735–736.

Desaulniers, G., J. Desrosiers, A. Erdmann, M. M. Solomon, and F. Soumis. 2001. "The Vehicle Routing Problem". Chapter VRP with Pickup and Delivery, edited by Paolo Toth and Daniele Vigo, 225–242. Philadelphia, PA, USA: Society for Industrial / Applied Mathematics. ISBN: 0-89871-498-2. `http://dl.acm.org/citation.cfm?id=505847.505856`.

Desrosiers, Jacques, François Soumis, and Martin Desrochers. 1984. "Routing with time windows by column generation". *Networks* 14 (4): 545–565. ISSN: 1097-0037. doi:`10.1002/net.3230140406`. `http://dx.doi.org/10.1002/net.3230140406`.

Dondo, Rodolfo, and Jaime Cerdá. 2007. "A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows". *European journal of operational research* 176 (3): 1478–1507.

Dorigo, Marco, Mauro Birattari, and Thomas Stutzle. 2006. "Ant colony optimization". *Computational Intelligence Magazine, IEEE* 1 (4): 28–39.

Dorigo, Marco, Vittorio Maniezzo, and Alberto Colorni. 1996. "Ant system: optimization by a colony of cooperating agents". *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26 (1): 29–41.

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

Dyckhoff, Harald. 1990. "A typology of cutting and packing problems". *European Journal of Operational Research* 44 (2): 145–159.

Eiben, Agoston E., and James E. Smith. 2003a. *Introduction to evolutionary computing.* 1st edition. Chapter 2, 10. Natural Computing Series. Springer.

———. 2003b. *Introduction to evolutionary computing.* 1st edition. Chapter 2. Natural Computing Series. Springer.

———. 2003c. *Introduction to evolutionary computing.* 1st edition. Chapter 10. Natural Computing Series. Springer.

Eilon, Samuel, and Nicos Christofides. 1971. "The loading problem". *Management Science* 17 (5): 259–268.

Fagerholt, Kjetil, Lars Magnus Hvattum, Trond AV Johnsen, and Jarl Eirik Korsvik. 2013. “Routing and scheduling in project shipping”. *Annals of Operations Research* 207 (1): 67–81.

Faroe, Oluf, David Pisinger, and Martin Zachariasen. 2003. “Guided local search for the three-dimensional bin-packing problem”. *Informs journal on computing* 15 (3): 267–283.

Fekete, Sandor P, and Jörg Schepers. 2000. “On more-dimensional packing I: Modeling”.

Fisher, Marshall L., Kurt O. Jörnsten, and Oli B. G. Madsen. 1997. “Vehicle Routing with Time Windows: Two Optimization Algorithms”. *Operations Research* 45 (3): 488–492. doi:`10.1287/opre.45.3.488`. eprint: `http://dx.doi.org/10.1287/opre.45.3.488`. `http://dx.doi.org/10.1287/opre.45.3.488`.

Fuellerer, Guenther, Karl F Doerner, Richard F Hartl, and Manuel Iori. 2010. “Metaheuristics for vehicle routing problems with three-dimensional loading constraints”. *European Journal of Operational Research* 201 (3): 751–759.

Funke, Birger, Tore Grünert, and Stefan Irnich. 2005. “Local search for vehicle routing and scheduling problems: Review and conceptual integration”. *Journal of heuristics* 11 (4): 267–306.

Garey, Michael R, and David S Johnson. 1979. *Computers and intractability.* Chapter 1, 2, 5, volume 174. freeman San Francisco.

Gendreau, Michel, Manuel Iori, Gilbert Laporte, and Silvano Martello. 2006. “A Tabu Search Algorithm for a Routing and Container Loading Problem”. *Transportation Science* (Institute for Operations Research)(the Management Sciences (INFORMS), Linthicum, Maryland, USA) 40, number 3 (): 342–350. ISSN: 1526-5447. doi:`10.1287/trsc.1050.0145`. `http://dx.doi.org/10.1287/trsc.1050.0145`.

Gilmore, Paul C, and Ralph E Gomory. 1963. “A linear programming approach to the cutting stock problem-Part II”. *Operations research* 11 (6): 863–888.

Glover, Fred. 1992. “New ejection chain and alternating path methods for traveling salesman problems”. In *Computer science and operations research,* 491–509. Elsevier.

Glover, Fred. 1996. “Ejection chains, reference structures and alternating path methods for traveling salesman problems”. *Discrete Applied Mathematics* 65 (1-3): 223–253.

Golden, Bruce L, Gilbert Laporte, and Éric D Taillard. 1997. “An adaptive memory heuristic for a class of vehicle routing problems with minmax objective”. *Computers & Operations Research* 24 (5): 445–452.

Golden, Bruce L, Subramanian Raghavan, and Edward A Wasil. 2008. *The vehicle routing problem: latest advances and new challenges.* Volume 43. Springer Science & Business Media.

Gürbüz, M Zahid, Selim Akyokuş, İbrahim Emiroğlu, and Aysun Güran. 2009. “An efficient algorithm for 3D rectangular box packing”.

Hung, Ming S, and J Randall Brown. 1978. “An algorithm for a class of loading problems”. *Naval Research Logistics Quarterly* 25 (2): 289–297.

Iori, Manuel, and Silvano Martello. 2010. “Routing problems with loading constraints”. *Top* 18 (1): 4–27.

Iori, Manuel, Juan-José Salazar-González, and Daniele Vigo. 2007. “An Exact Approach for the Vehicle Routing Problem with Two-Dimensional Loading Constraints”. *Transportation Science* 41 (2): 253–264. doi:10.1287/trsc.1060.0165. eprint: http://pubsonline.informs.org/doi/pdf/10.1287/trsc.1060.0165. http://pubsonline.informs.org/doi/abs/10.1287/trsc.1060.0165.

Irnich, Stefan, Paolo Toth, and Daniele Vigo. 2014. “Chapter 1: The Family of Vehicle Routing Problems”. Chapter 1 in *Vehicle Routing,* 1–33. doi:10.1137/1.9781611973594.ch1. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611973594.ch1. https://epubs.siam.org/doi/abs/10.1137/1.9781611973594.ch1.

Johnson, David S. 1973. “Approximation algorithms for combinatorial problems”. In *Proceedings of the fifth annual ACM symposium on Theory of computing,* 38–49.

Johnson, David S., Alan Demers, Jeffrey D. Ullman, Michael R Garey, and Ronald L. Graham. 1974. "Worst-case performance bounds for simple one-dimensional packing algorithms". *SIAM Journal on Computing* 3 (4): 299–325.

Kellerer, Hans, Ulrich Pferschy, and David Pisinger. 2004a. *Knapsack Problems.* Chapter 1. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-24777-7.

———. 2004b. *Knapsack Problems.* Chapter 15. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-24777-7.

———. 2004c. *Knapsack Problems.* Chapter 6. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-24777-7.

Knuth, Donald E. 1976. "Big omicron and big omega and big theta". *ACM Sigact News* 8 (2): 18–24.

Kohl, Niklas, and Oli Madsen. 1997. "An Optimization Algorithm for the Vehicle Routing Problem with Time Windows Based on Lagrangian Relaxation". *Operations Research* 45 (): 395–. doi:10.1287/opre.45.3.395.

Kohl, Niklas, and Oli B. G. Madsen. 1997. "An Optimization Algorithm for the Vehicle Routing Problem with Time Windows Based on Lagrangian Relaxation". *Operations Research* 45 (3): 395–406. doi:10.1287/opre.45.3.395. eprint: http://dx.doi.org/10.1287/opre.45.3.395. http://dx.doi.org/10.1287/opre.45.3.395.

Kolen, A. W. J., A. H. G. Rinnooy Kan, and H. W. J. M. Trienekens. 1987. "Vehicle Routing with Time Windows". *Operations Research* 35 (2): 266–273. doi:10.1287/opre.35.2.266. eprint: http://dx.doi.org/10.1287/opre.35.2.266. http://dx.doi.org/10.1287/opre.35.2.266.

Laporte, G., and F. Semet. 2001. "The Vehicle Routing Problem". Chapter Classical Heuristics for the Capacitated VRP, edited by Paolo Toth and Daniele Vigo, 109–128. Philadelphia, PA, USA: Society for Industrial / Applied Mathematics. ISBN: 0-89871-498-2. http://dl.acm.org/citation.cfm?id=505847.505852.

Laporte, Gilbert, Stefan Ropke, and Thibaut Vidal. 2014. “Chapter 4: Heuristics for the Vehicle Routing Problem”. Chapter 4 in *Vehicle Routing,* 87–116. doi:10.1137/1.9781611973594.ch4. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611973594.ch4. https://epubs.siam.org/doi/abs/10.1137/1.9781611973594.ch4.

Lawler, Eugene. 1985. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* New York: Wiley.

Lin, Shen. 1965. “Computer solutions of the traveling salesman problem”. *Bell System Technical Journal* 44 (10): 2245–2269.

Lodi, Andrea, Silvano Martello, and Michele Monaci. 2002. “Two-dimensional packing problems: A survey”. *European journal of operational research* 141 (2): 241–252.

Lodi, Andrea, Silvano Martello, and Daniele Vigo. 2002. “Heuristic algorithms for the three-dimensional bin packing problem”. *European Journal of Operational Research* 141 (2): 410–420.

Luke, Sean. 2013. *Essentials of Metaheuristics.* Second. Chapter 1. Lulu.

Mahvash-Mohammadi, Batoul. 2014. “Three-Dimensional Capacitated Vehicle Routing Problems with Loading Constraints”. PhD thesis, Concordia University.

Malapert, Arnaud, Christelle Guéret, Narendra Jussien, André Langevin, and Louis-Martin Rousseau. 2008. “Two-dimensional pickup and delivery routing problem with loading constraints”. In *First CPAIOR Workshop on Bin Packing and Placement Constraints (BPPC’08),* 184.

Männel, Dirk, and Andreas Bortfeldt. 2016. “A hybrid algorithm for the vehicle routing problem with pickup and delivery and three-dimensional loading constraints”. *European Journal of Operational Research* 254 (3): 840–858.

Martello, S, and P Toth. 1989. “An exact algorithm for the bin packing problem”. *EURO X, Beograd.*

Martello, Silvano, David Pisinger, and Daniele Vigo. 2000. “The three-dimensional bin packing problem”. *Operations Research* 48 (2): 256–267.

Martello, Silvano, David Pisinger, Daniele Vigo, Edgar Den Boef, and Jan Korst. 2007. “Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem”. *ACM Transactions on Mathematical Software (TOMS)* 33 (1): 7.

Martello, Silvano, and Paolo Toth. 1990a. *Knapsack problems: algorithms and computer implementations.* Chapter 1. John Wiley & Sons, Inc.

———. 1990b. *Knapsack problems: algorithms and computer implementations.* Chapter 8. John Wiley & Sons, Inc.

Martello, Silvano, and Daniele Vigo. 1998. “Exact solution of the two-dimensional finite bin packing problem”. *Management science* 44 (3): 388–399.

Massen, Florence. 2013. “OPTIMIZATION APPROACHES FOR VEHICLE ROUTING PROBLEMS WITH BLACK BOX FEASIBILITY”. PhD thesis, Louvain School of Engineering Louvain-la-Neuve Belgium.

Miao, Lixin, Qingfang Ruan, Kevin Woghiren, and Qi Ruo. 2012. “A hybrid genetic algorithm for the vehicle routing problem with three-dimensional loading constraints”. *RAIRO-Operations Research* 46 (1): 63–82.

Mittelmann, H. 2007. “Recent benchmarks of optimization software”. In *22nd Euorpean Conference on Operational Research.*

Motwani, Rajeev, and Prabhakar Raghavan. 2010. *Randomized algorithms.* Chapter 1,2. Chapman & Hall/CRC.

Parreño, Francisco, Ramón Alvarez-Valdés, JF Oliveira, and José Manuel Tamarit. 2010. “A hybrid GRASP/VND algorithm for two-and three-dimensional bin packing”. *Annals of Operations Research* 179 (1): 203–220.

Pisinger, David. 2002. “Heuristics for the container loading problem”. *European Journal of Operational Research* 141 (2): 382–392. ISSN: 0377-2217. doi:https://doi.org/10.1016/S0377-2217(02)00132-7. http://www.sciencedirect.com/science/article/pii/S0377221702001327.

Pisinger, David, and Mikkel Sigurd. 2007. “Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem”. *INFORMS Journal on Computing* 19 (1): 36–51.

Pollaris, Hanne, Kris Braekers, An Caris, Gerrit K Janssens, and Sabine Limbourg. 2015. “Vehicle routing problems with loading constraints: state-of-the-art and future directions”. *OR Spectrum* 37 (2): 297–330.

R. H. Mole, S. R. Jameson. 1976. “A Sequential Route-Building Algorithm Employing a Generalised Savings Criterion”. *Operational Research Quarterly (1970-1977)* 27 (2): 503–511. ISSN: 00303623. `http://www.jstor.org/stable/3008819`.

Reimann, Marc, Michael Stummer, Karl Doerner, et al. 2002. “A Savings Based Ant System For The Vehicle Routing Problem.” In *GECCO,* 1317–1326.

Ren, Jidong, Yajie Tian, and Tetsuo Sawaragi. 2011. “A relaxation method for the three-dimensional loading capacitated vehicle routing problem”. In *2011 IEEE/SICE International Symposium on System Integration (SII),* 750–755. IEEE.

Renaud, Jacques, Gilbert Laporte, and Fayez F Boctor. 1996. “A tabu search heuristic for the multi-depot vehicle routing problem”. *Computers & Operations Research* 23 (3): 229–235.

Rockafellar, RT. 2006. *Fundamentals of optimization.*

Ropke, Stefan. 2005. “Heuristic and exact algorithms for vehicle routing problems”.

Ropke, Stefan, and David Pisinger. 2006. “An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows”. *Transportation science* 40 (4): 455–472.

Rothlauf, Franz. 2011. “Optimization methods”. In *Design of Modern Heuristics,* 45–102. Springer.

Ruan, Qingfang, Zhengqian Zhang, Lixin Miao, and Haitao Shen. 2013. “A hybrid approach for the vehicle routing problem with three-dimensional loading constraints”. *Computers & Operations Research* 40 (6): 1579–1589.

Russell, Stuart J., and Peter Norvig. 2003a. *Artificial Intelligence: A Modern Approach.* 2nd edition. Chapter 3. Pearson Education. ISBN: 0137903952.

———. 2003b. *Artificial Intelligence: A Modern Approach.* 2nd edition. Chapter 4. Pearson Education. ISBN: 0137903952.

Schrijver, Alexander. 2005. "On the History of Combinatorial Optimization (Till 1960)". In *Discrete Optimization,* edited by G.L. Nemhauser K. Aardal and R. Weismantel, 12:1–68. Handbooks in Operations Research and Management Science. Elsevier. doi:http://dx.doi.org/10.1016/S0927-0507(05)12001-5. http://www.sciencedirect.com/science/article/pii/S0927050705120015.

Semet, Frédéric, Paolo Toth, and Daniele Vigo. 2014. "Chapter 2: Classical Exact Algorithms for the Capacitated Vehicle Routing Problem". Chapter 2 in *Vehicle Routing,* 37–57. doi:10.1137/1.9781611973594.ch2. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611973594.ch2. https://epubs.siam.org/doi/abs/10.1137/1.9781611973594.ch2.

El-Sherbeny, Nasser A. 2010. "Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods". *Journal of King Saud University - Science* 22 (3): 123–131. ISSN: 1018-3647. doi:http://dx.doi.org/10.1016/j.jksus.2010.03.002. http://www.sciencedirect.com/science/article/pii/S1018364710000297.

Simbolon, H, and H Mawengkang. 2014. "Mixed Integer Programming Model For The Vehicle Routing Problem With Time Windows Considering Avoiding Route". *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)* 3:2741–2744.

Taillard, Eric D, Gilbert Laporte, and Michel Gendreau. 1996. "Vehicle routeing with multiple use of vehicles". *Journal of the Operational research society:* 1065–1070.

Tarantilis, Christos D, Emmanouil E Zachariadis, and Chris T Kiranoudis. 2009. "A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem". *IEEE Transactions on Intelligent Transportation Systems* 10 (2): 255–271.

Thomas, Charles, Laurent Smet, and Yves Deville. 2015. “Local search for the vehicle routing problem”.

Toth, Paolo, and Daniele Vigo. 2001. *The Vehicle Routing Problem.* Edited by Paolo Toth and Daniele Vigo. Chapter 1. Philadelphia, PA, USA: Society for Industrial / Applied Mathematics. ISBN: 0-89871-498-2.

Toth, Paolo, Daniele Vigo, Paolo Toth, and Daniele Vigo. 2014. *Vehicle Routing: Problems, Methods, and Applications, Second Edition.* SIAM. ISBN: 1611973589, 9781611973587.

Vehicle Routing Problem. 2013. [Online; accessed 2018]. `http://neo.lcc.uma.es/vrp/vehicle-routing-problem/`.

Wäscher, Gerhard, Heike Haußner, and Holger Schumann. 2007. “An improved typology of cutting and packing problems”. *European journal of operational research* 183 (3): 1109–1130.

Wolpert, D. H., and W. G. Macready. 1997. “No Free Lunch Theorems for Optimization”. *Trans. Evol. Comp* (Piscataway, NJ, USA) 1, number 1 (): 67–82. ISSN: 1089-778X. doi:`10.1109/4235.585893`. `https://doi.org/10.1109/4235.585893`.

Wolsey, Laurence A, and George L Nemhauser. 1999. *Integer and combinatorial optimization.* Volume 55. John Wiley & Sons.

Yannakakis, Mihalis. 1990. “STACS 90: 7th Annual Symposium on Theoretical Aspects of Computer Science Rouen, France, February 22–24, 1990 Proceedings”. Chapter The analysis of local search problems and their heuristics, edited by Christian Choffrut and Thomas Lengauer, 298–311. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-46945-2. doi:`10.1007/3-540-52282-4_52`. `http://dx.doi.org/10.1007/3-540-52282-4_52`.

Zhang, Tao, Shanshan Wang, Wenxin Tian, and Yuejie Zhang. 2006. “ACO-VRPTWRV: A new algorithm for the vehicle routing problems with time windows and re-used vehicles based on ant colony optimization”. In *Sixth International Conference on Intelligent Systems Design and Applications,* 1:390–395. IEEE.

Zhao, Xiaozhou, Julia A Bennell, Tolga Bektaş, and Kath Dowsland. 2016. “A comparative review of 3D container loading algorithms”. *International Transactions in Operational Research* 23 (1-2): 287–320.