Isfandyar Khan Mian

Combining Vehicle Routing Optimization and Container Loading Optimization

Master's Thesis in Information Technology

March 25, 2020

University of Jyväskylä

Department of Mathematical Information Technology

Author: Isfandyar Khan Mian

Contact information: isfando@yahoo.com

Supervisors: Prof. Olli Bräysy, and 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 ans Service Engineering

Page count: 100+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 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. The combination of a Vehicle Routing and a 3D Container Loading problem creates new kinds of challenges. The problem was introduced in Gendreau et al. 2006a where the scalar capacity of the vehicles is replaced by 3D rectangular loading space. 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 identical 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. 2006a , 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 Martello, Pisinger, and Vigo 2000 as feasible. In order 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 which 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 vehicle's container in a fast but approximate way. In our experiments we use a 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 box is used for objects which are not rectangular in shape, such as cylindrical shapes. The LAFF algorithm places the boxes with largest surface area first by minimizing height from the bottom of the container. The box which covers the largest ground area of the container is placed first followed by subsequent boxes which 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 packed is small. During the LAFF stage, all real world bin packing constraints such as the weight of the boxes, the distribution of weight in the container, loading priorities, orientation, stacking, stability, etc. are ignored to gain as much speed as possible. After the LAFF stage, a more advanced algorithm will be used for the final packing of the orders, taking into account all constraints.

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. 2006a , 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 identtisiin, suorakaiteen muotoisiin kontteihin. Tämä ongelma on erittäin vaikea, koska se on yhdistelmä kahdesta edellä mainitusta ongelmasta, jotka molemmat ovat NP-vaikeita [Gendreau et al. 2006a , Pisinger 2002]. Eksaktin ratkaisun löytäminen tälle ongelmalle ei ole käytännöllistä, koska edes pelkän pienen konttien lastausongelman ratkaiseminen edellyttäisi enemmän laskentaresursseja Martello, Pisinger, and Vigo 2000 kuin mahdollista. 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. LAFF-vaiheen jälkeen lopullinen pakkaaminen suoritetaan käyttämällä edistyneempää algoritmia, joka ottaa huomioon kaikki rajoitukset.

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 almost one year old daughter, Zara Khan, for being a positive impact during this process through her joyous and motivating energy.

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 Hotaka for giving his opinions on certain topics in this thesis.

Jyväskylä, March 25, 2020

Isfandyar Khan Mian

List of Figures

List of Tables

Contents

1 Introduction

This introductory chapter provides the motivation, scope and research goals of the thesis. of this 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.

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 identical 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) or 3D Strip Packing Problem (3D-SPP), 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, the non-fragile items must not be placed over the fragile ones. 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 in to the container . The capacity checker would be fast which will introduce certain error in the process due to approximation of the current space in the container. Weight limitations of the container cannot cross 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 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, exact solution approach for realistic scenarios is not quite efficient for this problem yet. In order to solve 3 dimensional-Capacitated Vehicle Routing, the so-called Single Vehicle Loading Problem, needs to be solved in multiple times. In Kellerer, Pferschy, and Pisinger 2004 , it is discussed that the attempts to compute optimal solutions for the bin packing problem is not completely satisfying, especially if the available time and space is very limited. In the context of algorithmic performance time and and space are running time on computer and space is the amount of computer memory to solve a given problem respectively. It is clear that all algorithms which compute an optimal solution are not equivalent in their performance. It seems intuitive that an algorithm which computes approximate but not necessarily optimal solutions can reduce the drawback

of large computational time. It should also be noted that the difficulty level of algorithm also play 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. Still it is worth judging from the description of the algorithm whether they are straightforward or rather exhausting to implement.

So far, the best obtained solutions 3 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 Ant colony, Bee mating and hybrid solutions available in the literature. However, the overall process remains time consuming. The reason is that in a large Vehicle Routing Optimization problem 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 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 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 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.

2 Optimization

2.1 Introduction

The process of optimization is used to achieve the best possible results in a given situation (Astolfi 2006). In different fields such as design, construction and maintenance crucial decisions need to be made on day to day basis. The goal is to minimize effort and maximize benefit.

The effort or the benefit can be usually expressed as a function of certain design variables. Hence, opitimization is the process of finding the condition that gives the maximum or the minimum value of a function.

It is obvious that if a point x^* corresponds to the minimum value of a function $f(x)$, the same point corresponds to the maximum value of the function -$f(x)$. Thus, optimization can be taken to be minimization since every maximization problem can be reduced to a minimization problem.

2.1.1 Statement of an optimization problem

An optimization, or a mathematical programming problem can be stated as follows.

Find

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

which minimizes

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

subject to the constraints

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

and

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

The variable **x** is called the design vector, any system contains a set of quantities some of which are used as a variable in the design of optimization model. All the quantities that can be treated as variables are called design or decision variables, and are part of the design vector x.The variable $f(x)$ is the objective function, $g_j(x)$ are the inequality constraints and $l_j(x)$ are the equality constraints. The number of variables n and the number of constraints $p+m$ need not be related. If $p+m=0$, the problem is called an unconstrained optimization problem.There are other types of design constraints related to the 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 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 locally 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 5kg oranges and 2kg apples if price of one kg orange is 2 dollar and price of one kg apple is 5 dollar.

Range constraints The limitations which restricts the values of some decision variables to lie within specific closed intervals of $\mathbb{R}$. Range constraints are used to keep a variable between certain upper and lower bounds. An important example of this is the non negativity constraint in which some variable x_j may only be allowed to take values ≥ 0; with $[0,\infty)$ as interval. For example the speed of the car should be between the range of 80km/h to 100km/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 sum of constant coefficient times the variables $x_1, \ldots, x_n$.

Data parameters In addition to decision variables problem statement 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 classical design procedure aims at finding an acceptable design, i.e. a design which satisfies all the constraints.

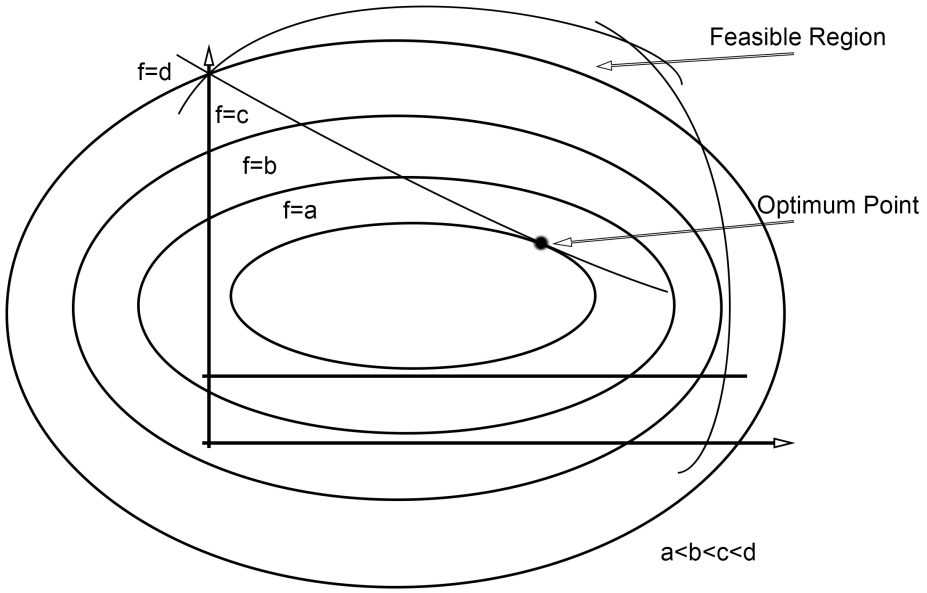


Figure 1: Design space, objective functions surfaces, and optimum point.

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 a criterion for comparing different designs. This criterion, when expressed as a function of the design variables, is known as objective function. The objective function is in general influenced by physical or economical considerations. However, objective function selection is not trivial, because what is the optimal design with respect to one criterion might be unacceptable with respect to another criterion. Typically, there is a trade off performance–cost, or perfor-

mance–reliability, hence the selection of the objective function is one of the most important decisions in the whole design process. In multiobjective optimization problem, the different criterion may be approximately solved considering a cost function which is a weighted sum of several objective functions.

Optimization problem can be classified in several ways:

- An optimization problem can be classified as a constrained or an unconstrained one, depending upon the presence or absence of constraints.
- Optimization problems can be classified as linear, quadratic, polynomial, non-linear depending upon the nature of the objective functions and the constraints. This classification is important, because computational methods are usually selected on the basis of such a classification, i.e. the nature of the involved functions dictates the type of solution procedure.
- Depending upon the values permitted for the design variables, optimization problems can be classified as integer or real valued, and deterministic or stochastic.

2.2 Asymptotic analysis

Asymptotic analysis of an algorithm, refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case and worst case scenario of an algorithm.

Following are commonly used asymptotic notations used in calculating running time complexity of an algorithm:

Big Oh Notation, O The $O(n)$ is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or longest amount of time an algorithm can possibly take to complete.
$O(f(n)) = g(n) : there exists\ c \geq 0\ and\ n_\theta\ such that\ g(n) \leq c.f(n)\ for all\ n > n_\theta.$

Omega Notation, ω The $\omega(n)$ is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or best amount of time an algorithm can possibly take to complete.

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

Theta Notation, θ The $\theta(n)$ is the formal way to express both the lower bound and upper bound of an algorithm's running time.

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

2.2.1 Computation Complexity

P Set of decision problems or class of problems for which some algorithm can solve the problem 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 running time of the algorithm for input size n.

$\exists$ a constant k such that the running time $T(n)$ is $O(n^k)$.

Example: sorting, minimum spanning tree.

NP In formal terms class NP can be defined as the set of decision problems where the "yes" instances can be decided in polynomial time by a non-deterministic Turing machine.

NP is the class of decision problems, for which the "yes" answers have proofs verifiable in polynomial time by a deterministic Turing machine. The notation NP stands for "nondeterministic polynomial time", since originally NP was defined in terms of nondeterministic Turing machines, that is machines having more than one possible move from a given configuration.

Example: Subset sum problem, bin packing problem.

Class P and Class NP can be represented as P and NP respectively. So, P is the class of "easy to solve" problems, and NP is the class of "easy to check" problems. The class P is contained in class NP i.e. $P \subseteq NP$. Does $P = NP$?

It is an open problem of major importance.

Is $P = NP$?

This is called the P vs NP problem. It is a major unsolved problem in the field of

computer science.

If $P = NP$, then it basically denotes the set of problems that can be verified in polynomial time (class NP) and can also be solved in polynomial time (Class P).

If $P \neq NP$, it means that there are problems in NP (quickly verifiable) that are hard to solve than to verify. This gives rise to the concept of Class NP-Complete and NP-hard problems.

NP-Hard The set of problems is said to be in NP-hard if it contains the following property

- If there exists a polynomial time algorithm to solve one of these problems then there exists one for every problem in NP.

Note: NP-hard problems need not be in NP and need not be a decision problem. A decision problem X is NP-complete if

- $X \in NP$

- X is NP-hard (or) if every problem in NP can be reduced to X in polynomial time.

X can be shown to be in NP by showing that a candidate solution to X can be verified in polynomial time.

NP-complete problems are the hardest problems in NP. The importance of solving a NP-complete problem is that if we are able to find an algorithm to solve NP complete problem in polynomial time then we can solve every other NP problem in polynomial time.

No efficient algorithm for an NP-complete problem has ever been found; but nobody has been able to prove that such as algorithm does not exist. For many NP optimization problems, serious attempts are made to find the optimal solution in polynomial time but since it appears to be intractable we limit ourselves to approximate solutions using r-approximate algorithms. They will discussed later on in this chapter.

2.3 Exact Methods

Exact algorithms are used to solve an optimization problem to optimality. Unless P = NP, such algorithm can't run in worst case polynomial time but researchers have been trying to find exact algorithms whose running time is exponential with a low base. As an example we would look into exact methods to solve the Vehicle Routing Problem with Time Windows (VRPTW).Vehicle Routing Problems are basically concerned with finding a way to visit a given set of customer locations using a given set of vehicles in such a way that a cost function such as total distance or total time, is minimized. Vehicle Routing Optimization belong to a class of problems called NP-hard. Solving NP-hard optimization problem is quite a challenging task for researchers since the beginning of computer history (long before the concept of NP-hardness was discovered) (Ropke 2005). Researchers have made significant progress recently in this field but for most of the problem only fairly small instance can be solved. Only moderately sized problems can be solved to optimality consistently. The exact methods for the VRPTW can be generally classified into three categories: Lagrange relaxation-based methods, column generation and dynamic programming. Also variants of integer programming can be used to solve optimization problems. Exact methods often perform very poorly (in some cases taking days or more to find moderately decent, let alone optimal, solutions even to fairly small instances) (El-Sherbeny 2010).

2.3.1 Lagrange relaxation-based methods

Different research papers have used different approaches to apply Lagrange relaxation-based methods e.g variable splitting followed by lagrange relaxation, K-tree approach followed by lagrange relaxation etc. A K-tree for a graph having $n+1$ nodes is a set of $n+K$ edges spanning the graph. In this context spanning the graph means traversing through all nodes of the graph with minimum possible number of edges. In Kohl and O. Madsen 1997 ,objective function states that costs should be minimized.

$$\min \sum_{k\in V}\sum_{i\in N}\sum_{j\in N} c_{ij}x_{ijk} \tag{2.1}$$

subject to following constraints,

$$\sum_{k\in V}\sum_{j\in N} x_{ijk} = 1, \forall i \in C \tag{2.2}$$

$$\sum_{i\in C} d_i \sum_{j\in N} x_{ijk} \leq q, \forall k \in V \tag{2.3}$$

$$\sum_{j\in N} x_{0jk} = 1, \forall k \in V \tag{2.4}$$

$$\sum_{i\in N} x_{ihk} - \sum_{j\in N} x_{hjk} = 0, \forall h \in C, \forall k \in V \tag{2.5}$$

$$\sum_{i\in N} x_{i,n+1,k} = 1, \forall k \in V \tag{2.6}$$

$$s_{ik} + t_{ij} - K(1 - x_{ijk}) \leq s_{jk}, \forall i, j \in N, \forall k \in V \tag{2.7}$$

$$a_i \leq s_{ik} \leq b_i, \forall i \in N, \forall k \in V \tag{2.8}$$

$$x_{ijk} \in \{0,1\}, \forall i, \in N, \forall k \in V \tag{2.9}$$

If constraint 2.2 is Lagrangian relaxed i.e. relaxing the constraint ensuring that each customer is served exactly once and the objective function can be written as

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

subject to 2.3 till 2.9

Here the lagrange multiplier λ_j which is associated with the constraint makes sure that the customer j is served

In Fisher, Jörnsten, and Madsen 1997 , two optimization methods are given for solving the Vehicle Routing Problem in an optimal fashion. The problem is formulated as a K-tree with degree 2K on the depot. The two methods are a K-tree relaxation with time windows added as a side constraint and a Lagrangian decomposition in which variable splitting is used to divide the problem into two subproblems. One will be a semi-assignment problem and the other a series of shortest path problems with time windows and capacity constraints.

2.3.2 Column generation

Column generation is an efficient algorithm for solving larger linear programs.The problem under consideration is split into two problems: the master problem and the subproblem. In the event that a linear program contains excessively numerous variables to be tackled explicitly, then we can introduce the linear program as master problem with a small subset of the variables and calculate a solution of this reduced linear program. Afterward, we check if the addition of one or more variables as subproblem, as of now not in the linear program, may enhance the linear program solution. This check should be possible by the calculation of the lessened costs of the variables. For this situation, a variable of negative lessened cost can enhance the arrangement.

In practical implementation a master problem is solved and the obtained solution is used to get dual prices for each of the constraints. This information is put to use in objective function of subproblem. Then the subproblem is solved. If objective value of the subproblem is negative, a variable with negative reduced cost is identified. This prompts the addition of variable to the master problem, and the master problem is re-solved. Re-solving the master problem repeats the intial cycle again until we obtain non-negative reduced cost. At this point we conclude that the solution to the master problem is optimal.

For the first time column generation to solve the VRP problem is used in Agarwal, Mathur, and Salkin 1989 , while Desrosiers, Soumis, and Desrochers 1984 used the column generation approach to solve the multiple traveling sales person (m-TSP) with time windows. This article used the column generation approach for the first time for solving the VRPTW.

2.3.3 Dynamic programming

Dynamic programming is a technique to solve complex optimization problems. The problem is broken down in to 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 sub problem 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 it will check the lookup table for precomputed solution before computing a solution.

The dynamic programming approach for VRPTW was presented for the first time in Kolen, Kan, and Trienekens 1987 . Christofides and Beasley 1984 used the dynamic programming paradigm to solve the VRP.

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 traverse branches of the tree, which represents 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 produce a better solution than the one that is already our 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 sub problems from further consideration is called bounding. Once a sub problem is removed from the tree in the bounding process, it will not be considered anymore and is not decomposed into further sub problems .

2.3.4 Integer Programming

In integer programming a mathematical optimization problem restricts all the variables to be integers. In most of the cases it can refer to Integer Linear Programs (ILP). MILP is a variant of integer programming problem and stand for Mixed integer linear programming (MILP). MILP involves problems in which only some of the variables are constrained to be integers, while other 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 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). In the transport of of hazardous material,

estimation and analysis of risks is an important aspect. One of the main focus is the selection of safest routes. In the first stage a linear approximation of the total routing risk is used as objective function. The routing risk measure is assumed as a non linear function of the truck load, which is approximated by means of two different piecewise linear functions (PLF). In the second stage to solve the overall risk optimization problem, mixed integer linear programming (MILP) is used integrate the best piecewise linear approximations of the routing risk. The MILP model can optimize risk for instances with 20 nodes in 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 problem with time windows and vehicles. When applied to VRPTW, their clustering algorithm could solve 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 problem 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 the the sub-route is avoided 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 utilised 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.

2.4 Approximation Algorithm

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

In principle , any algorithm that produce a good enough solution is an approximate algorithm. A feasible approximate algorithm would produce an approximate solution which is not too far from the optimal solution. In addition, we are interested to get some information about the size of this deviation from the optimal solution.

Let us take the example of bin packing which is NP-hard problem and the exact solution to bin packing problem takes exponential time. This increases the difficulty in finding 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 which is close to optimal solution and computed with the help of reasonable resources and time is considered a good solution .

In general, it may happen that an approximation algorithm sometimes produce solutions with an almost optimum value on specific data sets, but in other cases it will produce inferior solutions. In such situation, the worst case behavior of an approximation algorithm is studied. One way to measure the distance is absolution performance guarantee. It can be determined by the maximum difference between the optimal solution value and the approximate solution value over all problem instances. Suppose the optimal solution value of a problem instance I as $z^*(I)$ and the solution value computed by an approximation algorithm A as $z^A(I)$. when we are dealing with the approximation of problems where the goal is maximized the objective (knapsack) we have $z^A(I) \leq z^*(I)$.

Definition: An algorithm A is an approximation algorithm with absolute performance guarantee k, $k > 0$, if

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

holds for all problem instances I.

Another reasonable way to measure the distance between an approximate solution and an optimal one is the relative performance guarantee which basically bounds the maximum ratio between the approximate and an optimal solution. This expression of the distance as percentage of the optimal solution value seems to be more plausible than the absolute performance

guarantee since it is not dependent on the order of magnitude of the objective function value. 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 a 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.

2.4.1 Approximation Algorithm classes

NPO Class NPO can be defined as a set of problems that allows polynomial time r-approximate algorithm. The existence of r-approximate algorithm for NP-hard problems helps in finding the approximate or the closest possible solution to the optimal.

APX APX belongs the class of all NPO problems where for some $r \geq 1$ exista a r-approximate polynomial time algorithm.

So any problem which has r-approximate algorithm is said to be in class APX. Some of the problems which belong to the class APX are maximum satisfiability, maximum cut, minimum graph coloring restricted to planar graphs, minimum vertex cover, minimum bin packing and many more.

There are situations where for some NPO problems we cannot find r-approximate polynomial time algorithm unless P=NP, which can be interpreted as finding approximation algorithm is as hard as to determine optimal solution. This means that under the hypothesis $P \neq NP$, class APX is strictly contained in class NPO i.e. $APX \subseteq NPO$. Here in the given expression APX denotes class APX and NPO denotes class NPO.

Now as mentioned earlier in the above paragraph, we have situations where there are problems belonging to class NPO but does not belong to class APX. For example, minimum travelling salesperson problem is an optimization problem which does not have an r-approximate polynomial time algorithm. So it does not belong to class APX.

Some other problems which do not belong to class APX are maximum clique and maximum independent set problem. Unfortunately, for most of the problems in APX the performance ratio can only be approximated to a certain point, which means that a threshold exists t such that $r < t$ becomes computationally difficult.

PTAS Polynomial time Approximation Scheme

Let Q be an NP-hard optimization problem. An algorithm A is an approximation scheme for Q if for every $r > 0$, 'A'returns a solution Q_{sol} such that

$$Q_{sol} \leq (1+r)Q_{opt} \ldots \text{if Q is a minimization problem}$$
$$Q_{sol} \geq (1+r)Q_{opt} \ldots \text{if Q is a maxmization problem}$$

Q_{opt} means the optimal solution for the problem Q.

'A' will be called PTAS, if it runs in polynomial time of n and as we decrease r, the running time increases drastically. The dependency on r is exponential, so for example the running time can be of form $O(n^{\frac{1}{r}})$, $O(2^{\frac{1}{r}}n^3)$ and many more.

Now for any NPO problem, let us suppose there exists a constant k and if its NP-hard to describe that for a given instance I, $m_{OPT}(I) \leq k$, then there is no PTAS for that problem and a polynomial time algorithm with $r < \frac{k+1}{k}$ exists only if P=NP.

Class PTAS Class PTAS can be defined as the set of problems that allow PTAS or has a PTAS. So any algorithm which contains PTAS is said to belong in class PTAS.

By definition class PTAS belongs to class PAX. So the problem which does not belong to class APX does not have PTAS too. Example: minimum travelling salesperson problem. So if $P \neq NP$, then $PTAS \subset APX$ where PTAS represents class PTAS and APX denotes class APX respectively.

Bin packing problem does not have PTAS. If $P \neq NP$ and r is the approximation ratio to bin packing, there is no r- approximate polynomial time algorithm for minimum bin packing problem for which $r \leq \frac{3}{2} - \varepsilon, \varepsilon > 0$.

$PTAS^{\infty}$ Asymptotic Polynomial time Approximation Scheme is a weaker form of approximation when compared to PTAS. It is based on the idea that the performance ratio of

the approximate solution (returned by the respective approximation algorithm) may improve as optimal solution becomes bigger.

Just like class PTAS we also have class $PTAS^{\infty}$ which is the set of all NPO problems that contain an asymptotic polynomial time approximation ratio ($PTAS^{\infty}$). So the relation between PTAS, APX and polynomial time approximation ratio ($PTAS^{\infty}$) can be given as $PTAS \subseteq PTAS^{\infty} \subseteq APX$.

Let P be an NPO problem and let there exist a constant k. An algorithm A is said to be an asymptotic polynomial approximation scheme for any $r \geq 1$, if the algorithm A for the instance I returns a solution whose performance ratio is at most $r + \frac{k}{m_{OPT}(I)}$ where $m_{OPT}(I)$ denotes the optimal solution and algorithm A runs in polynomial time.

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 sufficient for the immediate goals. Heuristics can be any clever approach of 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 are deterministic algorithms that uses 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 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 represent 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 problem. A theorem named No Free Lunch 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 solution is critical. It can happen if an incident happens while following a certain plan or if a customer requirement for the transportation task changes. This is the reason fast construction algorithms are preferred for solving problems containing thousand of customers.

2.5.1 Neighbourhoods

The neighborhood of a solution S is a set N(S) of solutions that can be generated 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 is explored and a move to a new solution is made if and only if an improvement is made. In Vehicle Routing, neighborhoods problem can be divided into two major categories: 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. Single-Route neighborhoods will make a change to one route at a time i.e. they permute the customers within a route. Multi-Route neighborhoods exchange and move customers between two or multiple routes. This shows that Multi-Route neighborhoods can make more substantial changes 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 used in Laporte and Semet 2001 is used to describe a local search heuristic that only performs such moves that improve the value of

objective function of the solution. 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 for 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 sub field of stochastic optimization and combines a factor of randomness with heuristics. A metaheuristic refers to an iterative master strategy that guides and modifies the operations of subordinate heuristics by combining intelligently different concepts for exploring and exploiting the search space. It is useful specially when we have very little or incomplete information or limited computational power to know the global optimum solution to a problem in advance (Luke 2013). Metaheuristic is about optimizing the solution of a problem towards a better state but it does not guarantee to find the globally optimum solution. The advantage of metahueristic is the use of less computational power as compared to other optimization solutions.

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 Vehicle Routing the route construction heuristic work towards solution of the problem by inserting customers one at a time into partial routes until a feasible solution is obtained. Construction algorithms are mainly distinguished by the order in which customers are selected and by the method used 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 do not contain 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 introduction of new techniques such as metaheuristics.

Some examples are the PDPTW insertion heuristic by Lu and Dessouky 2005 , the VRPTW insertion heuristic by Ioannou, Kritikos, and Prastacos 2001 and the savings algorithm for the CVRP by İ. K. Altinel 2005 .

2.6 Evolutionary Algorithms

Evolutionary Algorithms consists of a collection of methods that have been originally developed to solve combinatorial optimization problems. They 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 same: given a population of individuals, the environmental pressure implements a procedure of natural selection, which causes the fitness of population to increase (Eiben and Smith 2003).

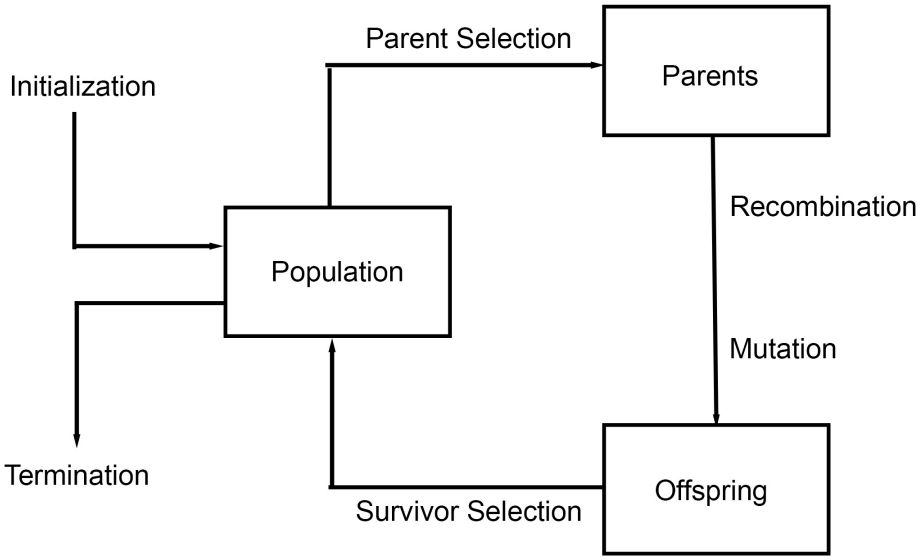


Figure 2: The general scheme of Evolutionary Algorithm as a chart.

Given a specific quality function which needs to be maximised, we randomly create a set of candidate solutions. We check the fitness of each candidate solution and the better candidates

are chosen to seed the next generation by applying recombination and/or 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 which competes with the generation of their parents solutions in fitness and possibly age. The process is repeated until a candidate with required fitness is found or the computational limit is reached.

Algorithm 1 General Scheme of Evolutionary Algorithm

```
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 evolutionary algorithm is to choose representation which suits our problem. In the next step we need to choose variation operators (recombination and mutation) to suit representation. The selection operators only use fitness and so are independent of representation.

2.6.1 Memetic Algorithm

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 area 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.

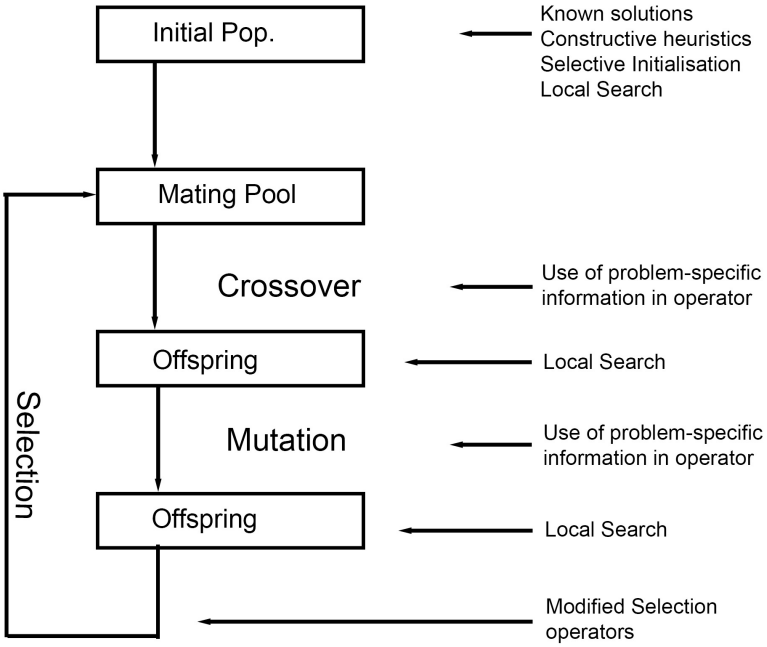


Figure 3: Flowchart of Memetic Algorithm.

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 is introduced and various techniques for solving the problem are discussed.

3.1 Loading Problem

The loading problem is about loading a set of items into a bin or container with the objective of minimizing the non utilized space in the container. The loading problem is addressed by Dyckhoff 1990 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. This paper Dyckhoff 1990 elaborates the logical structure and characteristics that need to be considered in the various kinds of packing problems.

The paper suggests a simple system of 96 (4 x 2 x 3 x 4) combined problems formed by combining some main types of the four important characteristics named dimensionality, kind of assignment, assortment of large objects (containers or vehicles) and assortment of small items. The detailed break down of the four characteristics is given in the list below. Each problem needs individual solution methods applicable to other types of problems only with major revision. Some the the related problem types are discussed below. In classical 1-dimensional knapsack there is one large object that has to be packed with a selection from the set of small items. In classical bin packing items are loaded in large objects are of the same figure. Pallet loading problem is about loading 2-dimensional congruent items into a large object. Two dimensional bin packing is concerned with packing the items into one object of given width and of minimal length (or height). The 1 dimensional vehicle loading problem is about packing the items into objects of identical figure. Container loading problem is 3 dimensional packing of items having length, width and depth into object with same figure.

1. Dimensionality
    - One-dimensional. (e.g cutting tubes)
    - Two-dimensional. (e.g cutting glass, cutting wood, 2D container loading)
    - Three-dimensional. (e.g 3D container loading)
    - N-dimensional with N > 3. (fourth dimension can be time)
2. Kind of assignment
    - All objects (containers) and a selection of items.
    - A selection of objects (containers) and all items.
3. Assortment of large objects (containers or vehicles)
    - One object.
    - Identical figure.
    - Different figures.
4. Assortment of small items (to be packed)
    - Few items (of different figures).
    - Many items of many different figures.
    - Many items of relatively few different (non-congruent) figures.
    - Congruent 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 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 transportation requests of the customers. Each request has a weight w_j with a corresponding profit p_j at per unit weight. The capacity of the container is represented by c.

The discussion below is inspired by Kellerer, Pferschy, and Pisinger 2004 . Suppose we have

n binary variables $x_j \in \{0,1\}$ corresponding to the selection in the *jth* binary decision and profit values by p_j indicating the difference of value attained by choosing between the two alternative values. After a suitable assignments of the two options to the two cases $x_j = 1$ and $x_j = 0$, we will always have $p_j \geq 0$. The total profit will be sum of n binary decisions. In this model we have decision problems where the feasibility of a particular set of alternatives selected can be evaluated by a linear combination of coefficients for each binary decision. The feasibility of a selection of alternatives is determined by a capacity restriction. Each binary decision j requires a weight or resource w_j if the first alternative $(x_j = 1)$ is chosen whereas selecting the second alternative $(x_j = 0)$ does not require a weight or resource. A selection of alternatives is feasible if the sum of weights of all the binary decisions does not surpass a given threshold capacity value c. This condition can be expressed as $\sum_{j=1}^{n} w_j x_j \; c$. The goal of this problem is to maximize the profit or value.

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.

The integer formulation of the knapsack is as following:

maximize

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

subject to

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

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

The knapsack problem has been studied for centuries and it is the simplest prototype of maximization problem.

The knapsack problem can be interpreted in different contexts. One such instance is an investment problem . An investor has specific amount of funds c available which he wants to invest in a profitable business. She would invest the money w_j and get expected 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 cutting problem. Assume that the a wooden log needs to be cut by a saw into small pieces. The pieces should have a predefined size represented by w_j having a selling price p_j. The length of the wooden log defines the capacity c of the problem.

3.3 Bin Packing Problem

The Bin-Packing Problem (BPP) can be formulated in the context of knapsack problems. We are given n items and n bins. An item j with a profit p_j and weight w_j. The capacity of each bin is c. The objective is to assign each item to one bin so that total weight of items does not exceed the bin capacity c and as few as possible bins are used. This can be formulated as

minimize

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

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

$$y_i = \begin{cases} 1 & \text{if bin i is used} \\ 0 & \text{otherwise} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{if item j is assigned to bin i} \\ 0 & \text{otherwise} \end{cases}$$

The weights w_j are assumed as positive integers. Hence, without losing the generality, we

will also assume

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

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

In case of violation of 3.6, c can be replaced by $\lfloor c \rfloor$. If an item violates assumption 3.7 i.e. the weight is more than the total capacity of bin, then the instance is trivially infeasible.

Offline algorithm 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 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 classified into three types: Single Bin Packing Problem (SBPP), Multiple Bin Packing Problem (MBPP), and Residual Bin Packing Problem (RBPP). The name residual depicts the remaining capacity of the bin or remain quantity after cutting a material. In SBPP, a set of different items must be assigned to a set of identical bins. In MBPP, a set of different items must be assigned to weakly heterogeneous bins in contrast to RBPP where the set of item must be assigned to a set of strongly heterogeneous bins. In all three types of bin packing, the objective is to minimize the number of bins.

3.4 Heuristics for Bin Packing

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

- Next Fit (NF) Algorithm: In Next Fit the items are placed in the order of their arrival. The next item is placed into the current bin if it fits. If it does not, that bin is closed and a new bin is started. The result of using this algorithm is shown in 4 . The result of the Next Fit algorithm according to our example is six bins and this result is clearly wasteful; however, it is acceptable if the information about free space in previous bins is not available.

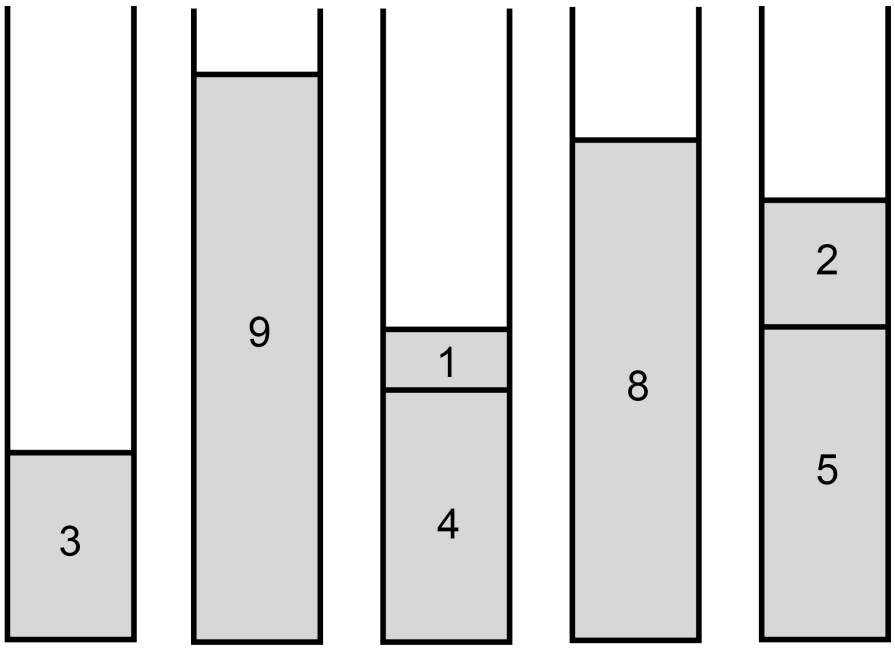


Figure 4: Packing Under Next Fit Algorithm .

- First Fit (FF) Algorithm: In First Fit the items are placed in the order of their arrival. The next item is placed into the lowest numbered bin in which it fits. If it does not fit into any of the currently opened bin, a new bin is started. The result of using this algorithm is shown in 5 . The result of the First Fit algorithm is five bins and this algorithm is dependent on keeping the previous bins in memory.

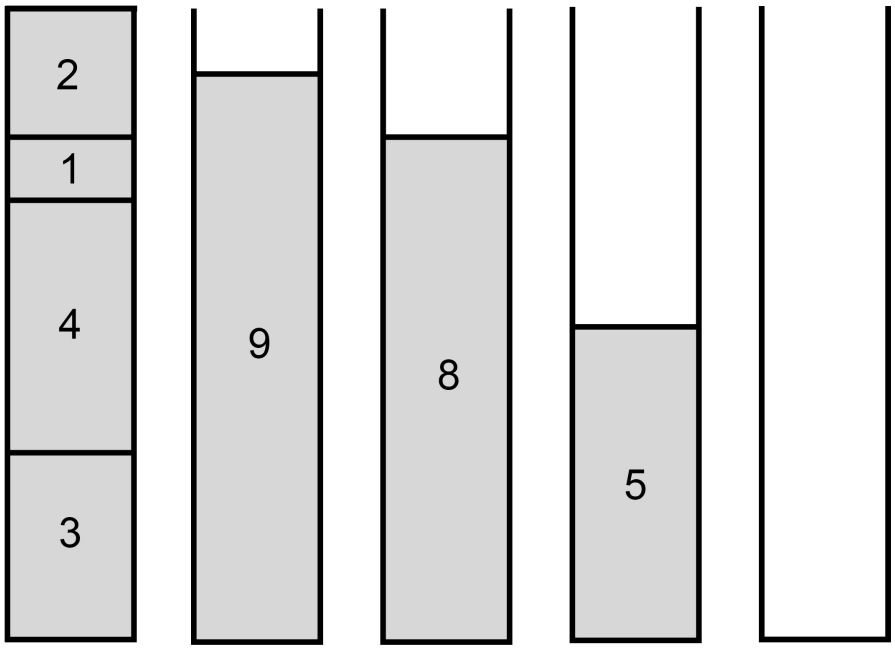


Figure 5: Packing Under First Fit Algorithm .

- Best Fit (BF) Algorithm: In Best Fit the items are placed in the order of their arrival. The next item is placed into that bin which will leave the least room left over after the item is placed in the bin. If the item does not fit in any bin, a new bin is started. The result of using this algorithm is shown in 6 . The result of the Best Fit algorithm is five bins as well and this algorithm is dependent on keeping a memory of previous bins. The Best Fit algorithm generally obtains the best solution in online algorithms.

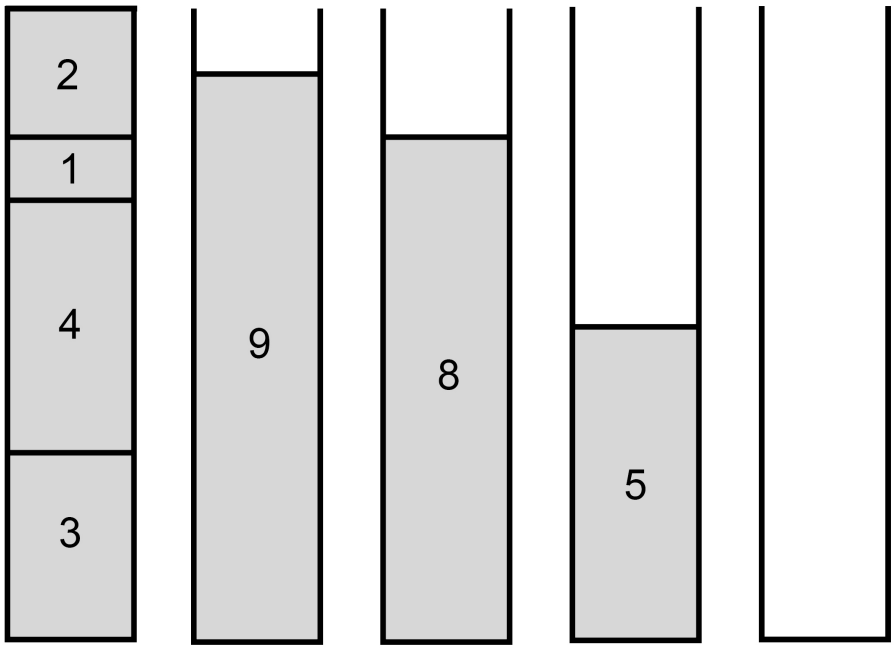


Figure 6: Packing Under Best Fit Algorithm .

- Worst Fit (WF) Algorithm: In Worst Fit the items are placed in the order of their arrival. The next item is placed into that bin which will leave the most room left over after the item is placed in the bin. If the item does not fit in any bin, a new bin is started. The result of using this algorithm is shown in 7 and the result of the Worst Fit algorithm is five bins. This algorithm is considered useful if all bins are desired to be the same weight approximately. It may not be useful for the upcoming items.

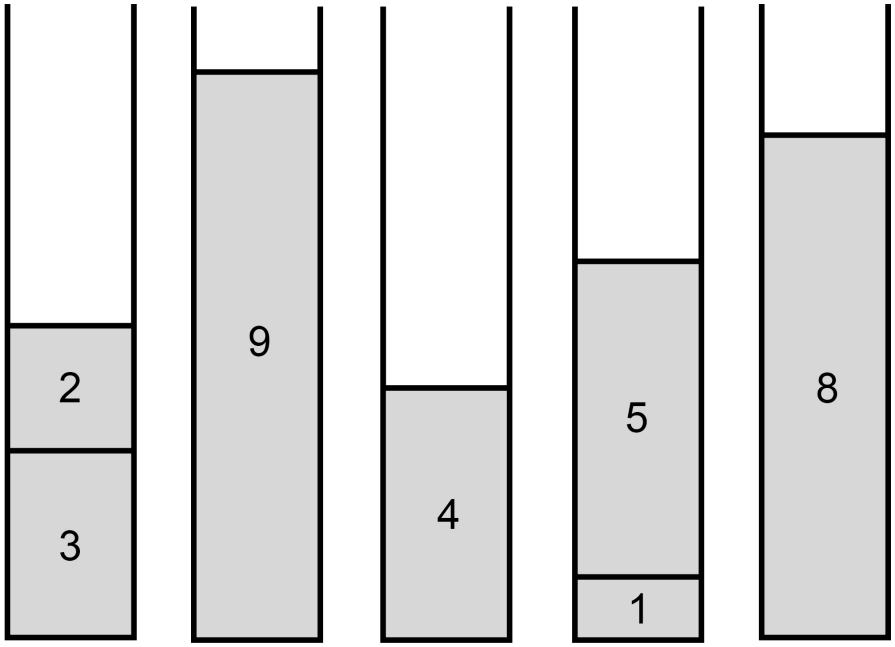


Figure 7: Packing Under Worst Fit Algorithm .

- First Fit Decreasing and Best Fit Decreasing (FFD-BFD) Algorithm: In First Fit Decreasing and Best Fit Decreasing the items are first sorted in decreasing order. In this case, the First Fit or Best Fit algorithm can be applied because they obtain the equivalent results. The result of using these algorithms is shown in 8 and the result of the (FFD-BFD) algorithm is four bins and the best result so far. These are offline algorithms because the information about items are available altogether from the beginning of the process.

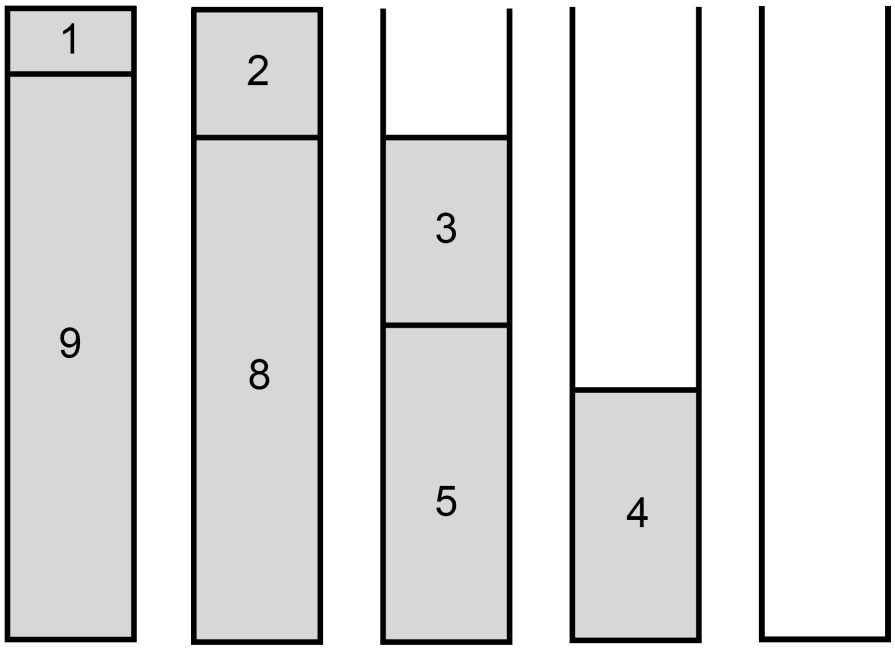


Figure 8: Packing Under First-Best Fit Decreasing Algorithm .

3.5 Exact Algorithms

A brief outline of exact algorithms is discussed based on Silvano Martello and Paolo Toth 1990. 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. It is a depth-first enumerative algorithm based on best fit decreasing branching. At any decision node, assuming that b bins have been initialized, let $(\bar{c}_{i_1} \ldots \bar{c}_{i_b})$ denote their current residual capacities sorted by increasing value, and $\bar{c}_{i_{b+1}} \equiv c_{b+1} = c$ the capacity of the next bin which is still un initialized: the branching phase assigns the free item j^* of largest weight, in turn, 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 accomodated. The branching strategy in this algorithm is based on characterization of equivilant 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 sorted according to their decreasing weight from the start. The bins are indexed according to the order in which they are initialized. At each decision node, the first largest free item is assigned, in turn, to the feasible initialized bins (by increasing index) and to a new bin. In the upcoming steps procedures are called on a node to fathom it and reduce 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 improvement of the current problem. A backtracking step happens in the form of removing the current item j^* from its current bin i^* and its assignment to the next feasible bin.

3.6 Approximate Algorithms

Bin packing is NP-hard problem and the exact solution to bin packing problem takes exponential time. This increases the difficulty in finding 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 which is close to 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 Silvano Martello and Paolo Toth 1990. A simpler approximate approach to bin packing is Next Fit (NF). In this algorithm the items are placed in the order of their arrival. The next item is placed into the current bin if it fits. If it does not, that bin is closed and a new bin is started. The complexity of Next Fit 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 satisfies the bound

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

here $z^*(I)$ denotes the optimal value of solution. Additionally, there exist instances for which the worst case performance ratio is $NF(I)/z^*(I)$ is close to 2 i-e r(NF)=2. As discussed earlier 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 called First Fit in which the items are placed in the order of their arrival. The next item is placed into the lowest numbered bin in which it fits. If it does not fit into any of the currently opened bin, a new bin is started. 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 exist instances I, with $z^*(I)$ arbitrarily large, 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 equation 3.2 so the worst-case performance does not provide full information on the worst-case behaviour. The asymptotic worst case performance ratio is commonly used 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,

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

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

The next algorithm is Best Fit in which the items are placed in the order of their arrival. The next item is placed into that bin which will leave the least room left over after the item is placed in the bin. If the item does not fit in any bin, a new bin is started. It has been proved in Johnson et al. 1974 that best fit has the same worst case bounds as first fit (as seen in 3.2 and 3.3), hence $r^{\infty}(FF) = \frac{17}{10}$. The time complexity of first fit and best fit is $O(n \log n)$.

The algorithms Next Fit Decreasing, First Fit Decreasing and Best Fit Decreasing are formed if the weights of the items are sorted in decreasing order first and then Next Fit, First Fit and Best Fit algorithm is applied. The worst-case analysis of Next Fit Decreasing has been found by Baker and Coffman 1981. The worst-case analysis of first fit decreasing and best fit decreasing are done in Johnson et al. 1974 proving the below equation for all instances I.

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

The rest of results are summarized in Table (taken from Coffman Jr, Garey, and Johnson 1984). 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
NFD	$O(n \log n)$	1.691...	1.424...	1.302 ...	1.234 ...
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

3.7 Two Dimensional Bin Packing Problem (2DBPP)

Two dimensional bin packing is concerned with packing a set of distinct rectangular items into rectangular bin without overlapping. The number of rectangular bins is assumed to be

unlimited. An item is specified by height and width (or length).

Gilmore and Gomory 1963 extended their 1BPP approach to model 2BPP 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. Sandor P 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 introduced an exact algorithm for two Dimensional finite bin packing problem. 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 takes the locally optimal choice at each node in a search space with the hope 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 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.
- Floor ceiling: Items are packed from left to right with their bottom edges on the level

floor (with respect to ceiling), and also from right to left with their top edges touching the level ceiling (with respect to floor), i.e., the horizontal line drawn on the top of the tallest item packed on the floor (Lodi, Martello, and Vigo 1999).

Lodi, Martello, and Vigo 1999 provides a good survey of the mathematical models, lower bounds, and greedy methods relevant to 2BPP and discusses heuristic and metaheuristic methods and exact approaches.

3.8 Three Dimensional Bin Packing Problem (3DBPP)

Two dimensional bin packing is concerned with packing a set of distinct rectangular items (having length, width and height) orthogonally in to rectangular bin (having length,width and height) without overlapping.

This discussion of various algorithms for 3DBPP is based on Mahvash-Mohammadi 2014. Chen, Lee, and Shen 1995 describes loading containers with cartons of non-uniform size and presents an analytical model to capture the mathematical essence of the problem. The 3DBPP problem is formulated as a zero-one mixed integer programming model with weight distribution and orientation constraints. To validate the above model and to explore its computational time, an example problem of three unequal-sized containers and six non-uniform cartons was solved with a LINGO (a 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 employed to verify the feasibility of packing items using concepts of corner points. Corner points are the point where a new item can accommodated within remaining empty space of the container in a given partial packing. Corner points reduce the number of partial solution explored. Two heuristic algorithms named as H1 and H2 are executed at each root node of the search tree. H1 builds a number of layers based on the depth of the bins and combines the layers into bins by solving a one dimensional bin packing problem defined on the depths of the layers . H2 minimizes the bins empty space by heuristically filling each one as full as

possible. The exact algorithm can solve instances with up to 90 boxes to optimality within a reasonable time 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 combining the original enumerative method with a new constraint-based programming approach. 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. The algorithm starts with an upper bound obtained on the number of bins through a greedy heuristic method. It iteratively tries to decrease this upper number by searching for a feasible packing of the boxes. The process stops when a given time limit is reached or the current solution is equal to precomputed lower bound. It has been tested on instances of up 200 boxes and results imply that produced solutions is equal to or 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. The heuristic method is based on layers, called height first area second (HA). The base of a bin is defined as the first layer. The floor of the first layer coincides with the base of a bin. The height of the tallest item available in the first layer is the floor of the second layer, and so on. HA selects the best solutions through the following two methods:

1. The items are partitioned into clusters characterized by non-increasing height, and a layered strip packing solution is determined. The obtained layers are then combined into finite bins through a 1DBPP algorithm.
2. The items are re-sorted by non-increasing area of their base and re-allocated to the current layers, possibly modifying 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 suggested by Martello, Pisinger, and

Vigo 2000. The average solution of Tabu Search is exactly the same as Faroe, Pisinger, and Zachariasen 2003 in one third of the instances. In some cases, Tabu Search is better that guides local search, and vice versa.

Crainic, Perboli, and Tadei 2008 introduced a new concept about extreme points (EPs). To better utilize the bin's volume, the concept of extreme points (EPs) is introduced. EPs are extension of CPs. Using EPs, a new heuristic algorithm is developed based on the first fit decreasing and the best fit decreasing. Crainic, Perboli, and Tadei 2009 also proposed a two-level Tabu Search called TS2PACK for 3DBPP. In TS2PACK, the first-level aims to reduce the number of bins and the second optimizes the packing of the bins. This second procedure is based on the Interval Graph representation of the packing, proposed by Sándor P Fekete and Schepers 1997, which reduces the size of the search space. The k-chain-moves procedure is also applied to maximize the neighborhood and to improve the quality of the solutions. Extensive Computational results on benchmark problem instances show that the proposed approach gives better results than existing ones.

A greedy randomized adaptive search procedure (GRASP) for 3DBPP is proposed by Parreño et al. 2010. The GRASP algorithm iteratively combines a constructive procedure and an improvement phase in order to obtain an efficient solution. The constructive phase is based on a maximal-space heuristic developed for the container loading problem. In the improvement phase, several new moves are designed and combined in a Variable Neighborhood Descent(VND) structure. In the improvement phase, various moves in a Variable Neighborhood Descent (VND) structure are used to improve the solution obtained by the constructive procedure. GRASP/VND algorithm gives solutions equal to or better than Crainic, Perboli, and Tadei 2009

4 Vehicle Routing Problems

A generic verbal definition Toth et al. 2014 of the family of Vehicle Routing Problems can be the following:

> **Given:** A set of *transportation requests* and a *fleet of vehicles*.

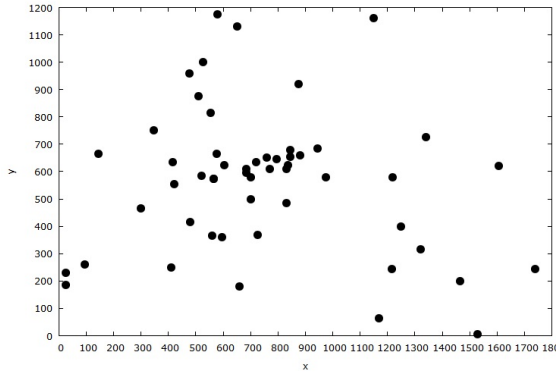
The problem is then to find a plan for the following:

> **Task:** Determine a set of *routes* to perform the transportation requests with the given fleet of vehicles *at minimum expenditure*; in particular, decide which *vehicle handles which customer requests in which sequence* so that service to all *routes* can be *feasibly* provided.

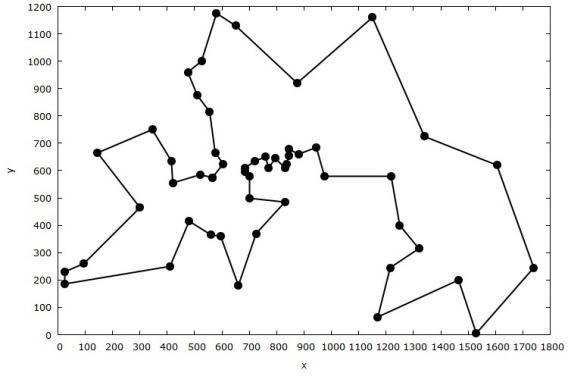
The original Vehicle Routing Problem has been stated in Dantzig and Ramser 1959. The problem was about distributing gasoline to different service stations around the city. The authors of that paper want to find out the optimum routes for the oil delivery trucks. Each truck contains a fuel tanker. The goal is to use least possible trucks that are driving 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 complicated variants of the problem still constitute a challenge for the researchers. This chapter provides the background, description and various methods to solve the Vehicle Routing Problem.

4.1 The traveling salesman problem

The traveling salesman problem is a touring problem in which each city must be exactly visited once. Different unvisited cities are shown in figure 9a while the figure 9b shows a single tour visiting all the cities once. The goal is to find the shortest possible tour while satisfying the constraint mentioned earlier Russell and Norvig 2003. The problem is NP-hard but researchers have given it a lot of emphasis due to its application in real life problem like automatic circuit board drill, stocking machines on shop floor and vehicle routing problems.



(a) Cities Nonvisited



(b) Cities Visted

Figure 9: TSP illustration.

There are different variants of traveling salesman problem (Ropke 2005). In symmetric variant for all cities, x and y the distance from city x and y is the same as the distance from city y and x. In asymmetric variant for all cities, x and y the distance from city x and y is not the same as the distance from city y and x. In Euclidean variant the cities must be located in $\mathbb{R}^d$and the distance between the cities is euclidean distance.

The traveling sales man problem is defined on a directed graph $G = (V,A)$ representing the arc network. The set of vertexes $V = \{1,\ldots,n\}$ represents the different locations and the set of arcs A is the set of directed edges. The figure 10 show an example of traveling sales man problem with a directed graph.

For each $(i, j) \in A$ is assigned a distance or cost c_{ij}. A binary decision variable x_{ij} is set to one if and only if arc (i, j) is used in the solution. The problem can be formulated as

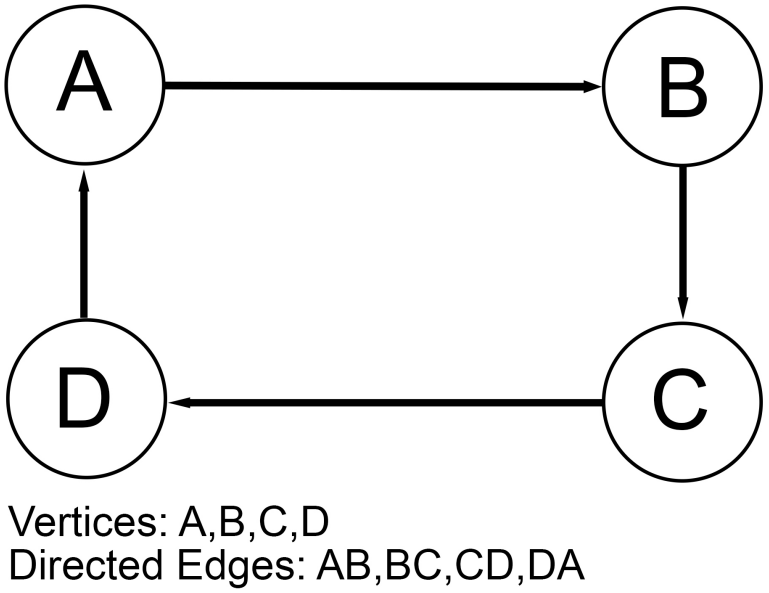


Figure 10: Travelling sales man directed graph

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

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 4.1 minimizes the arc costs, equations 4.2 and 4.3 ensures that one arc leaves each node and one arc enters each node, equation 4.4 eliminates sub-tours. Sub tours are multiple tour than one big tour through all the points.

There is a vast amount of literature present on the traveling salesperson problem. For a

start, Lawler 1985 is a good resource. This historic origin of the traveling salesman problem is discussed in Schrijver 2005 . The traveling salesman problem is the parent problem of vehicle routing problem. It is analogous to a vehicle routing problem with a fleet of one vehicle.

4.2 m-Traveling salesman problem

The m-Traveling Salesman Problem (m-TSP) is a generalization of the traveling salesman problem that consists of more than a salesman. In m-TSP we are given x cities, m salesman and a depot. Every city should be visited only once 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 determine a tour for each salesman such that the total tour cost is minimized. If distances satisfy the triangle inequality, i.e. if $d(i,k) \leq d(i,j) + d(j,k)$ for all *i, j* and *k* then we can easily confirm that the distance of the shortest TSP tour on the x cities plus the depot is always less than or equal to the distance of the shortest m-TSP solution for any m. The total distance of the m-TSP solution is the distance of all selected tours.

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 that his initial depot. The initial number of salesman on each depot must remain the same after all the trips are finished.

Number of salesmen 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 non fixed salesman scenario a cost is attached to 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 on 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

Vehicle Routing Problems are basically concerned with finding a way to visit a given set of customer locations using a given set of vehicles in such a way that a cost function is minimized. The cost function is generally taken as total distance covered by all the vehicle to interact with all customers and return to its 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 is described below.

Customers Customers are the location that are visited for the purpose of picking up or delivering goods. They represent real geographical locations on the map.

Vehicles Vehicles are the objects that are used to transfer the good to or from customers. There can be different attributes for the vehicle in various real life scenario. 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 real geographical locations that is used to store goods. Generally in vehicle routing problems, depot represents the home for vehicles. Vehicle start 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 road network is represented by a graph where customers and depots are represented by vertices. 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 of sequential locations that are visited by one vehicle in 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 depot. There are two types of fleets i.e. homogeneous fleet and heterogeneous fleet. In homogeneous fleet, all the vehicles have the same capacity and performance while in heterogeneous fleet, the performance and capacity in a set of the 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 a set of routes in which each route is traveled by one vehicle from the given vehicle set.

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

4.4 The Capacitated VRP

The Capacitated Vehicle Routing Problem is the most basic category among different variants of the vehicle routing problems. In this section we describe the problem and give the relevant notations. The idea for the structure of notation is taken from Massen 2013.

4.4.1 Problem Description

The Capacitated Vehicle Routing Problem is about delivering service to a set of customers with associated demands using a given set 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 does not exceed the threshold limit of the truck.

The problem is defined on a simple, complete and weighted graph $G = (V,A)$ representing the road network. The set of vertexes $V = \{0,\ldots,n\}$ represents the different locations and

the set of arcs A a sequence of roads to get from one location to another in the network. The set of outgoing (incoming) arcs from vertex $i \in V$ is denoted by $\delta_i^-(\delta_i^+)$. Vertexes $i = 1,\ldots,n$ are called customers, while vertex 0 is called depot. The weight c_{ij} of an arc $(i,j) \in A$ equals the distance on the road network between the locations matching vertexes i and j and thus represents the cost of including this arc in a solution. The distances are considered to be symmetric ($c_{ij} = c_{ji} \ \forall \ i,j \in V$) and to respect the triangle inequality ($c_{ij} \leq c_{ip} + c_{pj} \ \forall \ i,j,p \in V$). With each customer $i \in V \setminus \{0\}$ is associated a demand q_i. A set of homogeneous vehicles K, each of limited capacity Q, is available to perform the visits to the customers. Each vehicle may execute at most one route. Each route starts from the depot to visit a number of customers and then returns back to the depot. Since by definition the first and last vertexes in a route are the depot vertex, a route r can be seen as a tuple (S,σ) where $r.\sigma = \langle e_1,\ldots,e_m\rangle$ $(\cap_{i=1}^{m}\{e_i\} = \varnothing)$ is a sequence of customers $(\{e_1,\ldots,e_m\} \subseteq V \setminus \{0\})$ and $r.S = (e_1,\ldots,e_m)$. The customer visited in the s^{th} $(1 \leq s \leq |r.S|)$ position of route r is given by $r[s]$. For any route r , $r[0]$and $r[|r.S|+1]$represent the depot. Note that if for any route r we have $r[s] = i$ and $r[s+1] = j(0 \leq s \leq |r.S|)$, this means that arc (i,j) is used in this route.

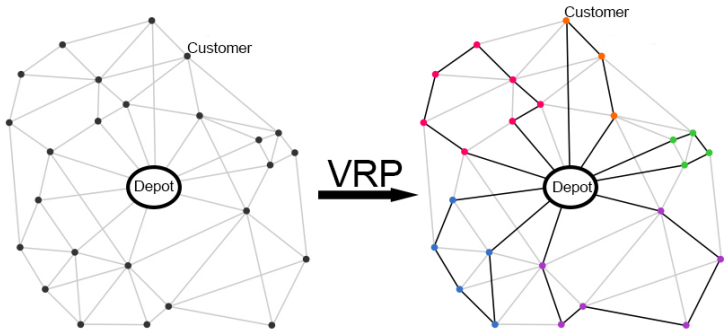


Figure 11: Vehicle Routing Problem

The intention is to make a solution $Sol = \{r_1 \ldots r_m\}$ containing a set of routes such that:

1. $|Sol| \leq |K|$

the solution does not use more vehicles than available

2. $(\cap_{ri \in Sol}\ r_i.S = \varnothing)$ and $\bigcup_{ri \in Sol}\ r_i.S = V \backslash \{0\}$
   to make sure that each customer is visited exactly once for delivering goods
3. $\sum_{j \in r.S}\ q_j \leq Q \quad \forall\, r \in Sol$
   the total volume for a route should not exceed vehicle capacity Q
4. min $\quad Dist(Sol) = \sum_{r \in Sol}\ (\sum_{S=0}^{|e.S|}\ c_{r[S]r[S+1]})$ minimized total distance

A solution complying with constraints 1-3 is called feasible. A solution partially or totally not complying with the constraints is called infeasible. The quality of a solution *Sol* is evaluated using $Dist(Sol)$. 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 which has a higher quality than the solution under consideration. Deciding whether a feasible solution for a given CVRP instance exists is NP-complete, while the problem of finding the optimal solution is NP-hard (Toth et al. 2014).

4.4.2 Capacitated VRP Variants

There are a number of variants of CVRP discussed in Golden, Raghavan, and Wasil 2008 and Toth et al. 2014. The types of modifications that give birth to a new CVRP variant along with examples is given below.

1. Changes to routes structure

   - **Multi-depot Vehicle Routing Problem**
     Instead of a single depot vehicles start and end their routes at different depots, the resulting problem is known as the Multi(ple) Depot VRP in Renaud, Laporte, and Boctor 1996.
   - **Capacitated Vehicle Routing Problem With Split Delivery**
     In some scenarios if a customer demand exceeds the vehicle capacity then service can't be provided in one visit. On the other hand, splitting services into several smaller service requests can be beneficial in overall cost savings. The Split Delivery VRP (Dror and Trudeau 1990) allows, in principle, that each demand be split into arbitrarily many smaller demands served by different vehicles.

- **Capacitated Vehicle Routing Problem With Multiple Trips**

  Generally for many VRP variants each vehicle performs only one route. In the VRP with multiple trips (Taillard, Laporte, and Gendreau 1996) , vehicles may perform several routes.

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 above situation it is better to provide service to a subset of customers. The goal is to maximize profit, defined as difference between the profit collected at visited customers and the total distance covered by fleet (Archetti, Speranza, and Vigo 2014).

   - **MinMax Vehicle Routing Problem**

     Instead of minimizing the total distance we may apply a min-max objective, e.g., in order to minimize the length (or duration, or workload) of the longest route (Corberán and Laporte 2015).

   - **Vehicle Routing Problem with Minimization of 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 hierarchical way of optimizing is to minimize the number of vehicles first and then, with this fixed a secondary objective such as the total distance of the trips is minimized (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 contraint i.e. requiring the consideration of travel, service and waiting times together with time-window constraints. The service to each customer should be provided within the customer's time window. It is possible for vehicle to wait if they arrive before the start of customer's time windows. The service time at 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 with Pick-up and Delivery**

  Pickup-and-delivery problems are vehicle routing problems in which the transportation requests consist of point-to-point transports. Each customer request is of the movement of goods or people between two particular locations, one where someone or something is picked up, and a corresponding location for the delivery. This problem also introduces precedence constraint between the pick-up and delivery location. The pick up location should be visited before the delivery location for a respective customer request. The loading space of vehicle will not steadily decrease or increase during the execution of route (Desaulniers et al. 2001).

- **Capacitated Vehicle Routing Problem with Loading Constraints**

  Complex loading constraints can occur when both the pellet and the cargo compartments are described either by 2-dimensional or 3-dimensional quantities. In the CVRP with 2-dimensional Loading constraints (Iori, Salazar-González, and Vigo 2007), shipments are rectangular items that have to be feasibly assigned to a rectangular compartment. Delivery of items to the same customer should be done by the same vehicle (item clustering constraint). The orientation of items during delivery may or may not be changed (item orientation constraint). When delivering an item to a customer, no items of other customers served later along the route may lay, not even partially, in the rectangular area between that item and the door of the vehicle (sequential loading constraint). As in the CVRP, a capacity constraint with respect to the weight (kg) has to be taken into account as well. The problem with 3-dimensional Loading constraints as described in Gendreau et al. 2006a is to ensure the stability of stacked boxes, the secure transportation of fragile boxes, and the easy unloading of boxes at the customer locations.

4.5 Notations and Operations on routes

As mentioned in 4.4.1 a route r can be seen as a tuple (S, σ) where $r.S$ represents the set of customer visited by r and (r, σ) the sequence in which they are visited.

- **Position of customers in a route**

For each route r , $r.\sigma = \langle e_1, \ldots, e_m \rangle$ represents the sequence in which customers in $\{e_1, \ldots, e_m\}(r.S)$ are visited. The vertex visited in the s^{th} position (i.e. e_s , $1 \leq s \leq |r.S|$) can be obtained by $r[s]$, i-e $r[s] = e_s$. Also $r[0]$ and $r[|r.S|+1]$ default to 0, the depot vertex. We can also state that the postion of a customer v in r s.t. $v \in r.S$ (the position corresponds to index s s.t. $e_s = v$) is obtained via $pos(v,r)$. Thus $r[pos(v,r)] = v \; \Lambda \; e_{pos(v,r)} = v \; \forall \; v \in r.S$. Finally, the first and last customers visited in r are retrieved using $first(r)$ and $last(r)$. If $|r.S| \geq 1$, $first(r) = r[1]$ and $last(r) = r[|r.S|]$, else if $|r.S| = 0$ the route is "empty" and $first(r) = last(r) = 0$.

Example: Let route r_{xx} where $r_{xx}.S = \{v_a, v_b, v_c, v_d\}$ and $r_{xx}.\sigma = \langle v_a, v_b, v_c, v_d \rangle$. The third customer in the route is $v_c, r_{xx}[3] = v_c$ and $pos(v_b, r_{ex}) = 2$. Furthermore $r_{xx} = 0$ and $r_{ex}[4+1] = 0$. Also $first(r) = v_a$ and $last(r) = v_d$.

- **Total demand of a route**

  For a route r the accumulated demand on r is given by $demand(r) = \sum_{i \in r.S} q_i$.

- **Distance of a route**

  For a route r the distance of r is given by $distance(r) = \sum_{i=0}^{|r.S|} c_{r[i]r[i+1]}$.

- **Operations on routes**

  A lot of vehicle route optimization approaches modify the routes in one way or another. The basic operations are given below

  - *Inserting a customer in a route*

    Consider a route r with $r.\sigma = \langle e_1, \ldots, e_{|r.S|} \rangle$. The insertion of customer $i (i \notin r.S \; \Lambda \; i \neq 0)$ in route r at position $p (1 \leq p \leq |r.S|+1)$ is denoted by $insert(i,r,p)$. Let r be the original route and $r^{'} = insert(i,r,p)$. Then $r^{'}.S = r.S \bigcup \{i\}$ and $r^{'}[t] = r[t] \; \forall \; 1 \leq t < p$ and $r^{'}[p] = i$ and $r^{'}[h] = r[h-1] \; \forall \; p+1 \leq h \leq |r^{'}.S|$. Analogously $r^{'}.\sigma = \langle r.\sigma_1^{p-1}, i, r.\sigma_p^{|r.S|} \rangle$ where $r.\sigma_i^t$ denotes the possibly empty sub-sequence $\langle r_i, \ldots, e_t \rangle$ of $r.\sigma$.

    Example: Let route r_{xx} where $r_{xx}.S = \{v_a, v_b, v_c, v_d\}$ and $r_{xx}.\sigma = \langle v_a, v_b, v_c, v_d \rangle$,the insertion of i in the second position in r_{xx} results in $r_{xx}^{'} = insert(i, r_{xx}, 2)$ where $r_{xx}^{'}.S = \{v_a, v_b, v_c, v_d, i\}$ and $r_{xx}^{'}.\sigma = \langle v_a, i, v_b, v_c, v_d \rangle$.

  - *Removing a customer from a route*

The removal of a customer i ($i \in r.S$) from route r is denoted by $remove(i,r)$. Let r be the orignal route and $r^{'} = remove(i,r)$. Then $r^{'}.S = r.S \backslash \{i\}$ and $r^{'}[t] = r[t]\ \forall\ 1 \leq t < pos(i,r)$, and $r^{'}[h] = r[h+1]\ \forall\ pos(i,r) \leq h \leq |r^{'}.S|$. Analogously $r.\sigma^{'} = \langle r.\sigma_{1}^{pos(i,r)-1}, r.\sigma_{pos(i,r)+1}^{|r.S|} \rangle$.

Example: The removal of v_b from route r_{xx} results in $r_{xx}^{'} = remove(v_b, r_{xx})$ where $r_{xx}^{'}.S = \{v_a, v_c, v_d\}$ and $r_{xx}^{'}.\sigma = \langle v_a, v_c, v_d \rangle$.

– *Merging two routes*

Consider two routes r_1 and r_2 such that $r_1.S \bigcap r_2.S = \varnothing$ and where $r_1.\sigma = \langle e_{11}, \ldots, e_{1|r_1.S|} \rangle$ and $r_2.\sigma = \langle e_{21}, \ldots, e_{2|r_2.S|} \rangle$. Merging r_1 with r_2 is denoted by $r = r_1\, X\, r_2$ where $r.S = r_1.S \bigcup r_2.S$ and $r.\sigma = \langle e_{11}, \ldots, e_{1|r_1.S|}, e_{21}, \ldots, e_{2|r_2.S|} \rangle$. Analogously $r[i] = r_1[i]\ \forall\ i \in 1, \ldots, |r_1.S|$ and $r[j + r_1.|S|] = r_2[j]\ \forall\ j \in 1, \ldots, |r_2.S|$. It should be noted that the merge operator X is not commutative, i.e. $r_1 X r_2 \neq r_2 X r_1$.

Example: Given two routes route r_x and r_y where $r_x.S = \{v_a, v_b, v_c\}$, $r_x.\sigma = \langle v_a, v_b, v_c \rangle$ and $r_y.S = \{v_g, v_h, v_i\}$, $r_y.\sigma = \langle v_g, v_h, v_i \rangle$ the merge r_x and r_y results in $r = r_x\, X\, r_y$ where $r.S = \{v_a, v_b, v_c, v_g, v_h, v_i\}$ and $r.\sigma = \langle v_a, v_b, v_c, v_g, v_h, v_i \rangle$. The merge of r_x and r_y results in route $r^{'} = r_y$ and r_x where $r^{'}.S = \{v_a, v_b, v_c, v_g, v_h, v_i\}$ and $r^{'}.\sigma = \langle v_g, v_h, v_i, v_a, v_b, v_c \rangle$.

4.6 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 do not contain 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 goal is to keep the distance of the solution as small as possible in case we are minimizing the total distance. Route construction heuristic work towards solution of the problem by inserting customers one at a time into partial routes until

a feasible solution is obtained. Some of the commonly encountered construction heuristics for CVRP are given below.

4.6.1 Savings Heuristic

The Clarke and Wright algorithm (Clarke and Wright 1964) is a widely known heuristic based on the notion of saving. In the paper, a sequential and parallel version are proposed. We will discuss the parallel version as it is appears to better performing (Laporte and Semet 2001), the feasible route merger yielding the largest saving is implemented at each iteration, until no more merger is feasible as shown in the figure 12.. 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 $r_1 = (0,\ldots,i,0)$ and $r_2 = (0,j,\ldots,0)$ can be feasibly merged into a single route $(0,\ldots,i,j,\ldots,0)$, a distance saving $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ is generated. The heuristic then puts every single customer in a route of its own, such that we have $Sol = r_1,\ldots,r_n (n = |V\backslash\{0\}|)$ and $r_i.S = \{i\}(1 \leq i \leq n)$. As a next step the routes in Sol are being merged. At each iteration the ordered pair (i,j) (where $(i,j \in V\backslash\{0\}\,,\, i \neq j)$) maximizing s_{ij} and such that $\exists r_a, r_b \in Sol$ with $last(r_a) = i$ and $first(r_b) = j$ and $first(r_b) = j$ and with $demand(r_a) + demand(r_b) \leq Q$ is determined. Route r_a is then merged with route r_b. Hence a new route $r^{'} = r_a X\, r_b$ is created and used to replace r_a and r_b in Sol. The new Sol with a reduced set of routes $Sol = (Sol\backslash\{r_a, r_b\}) \cup \{r^{'}\}$.

This heuristic continues merging routes until no more merging is possible. The set of routes remaining after execution of the heuristic is our final solution. We can use this solution if we have freedom of using vehicles as per the final routes of our solution. If the amount of our vehicles is fixed and it is less than the routes formed (i.e. $|Sol| > |K|$), the solution is infeasible.

4.6.2 Insertion Heuristics

Insertion heuristics develops a solution by inserting one customer after another into open routes. The term "open" route is used to denote route where customers can still be inserted.

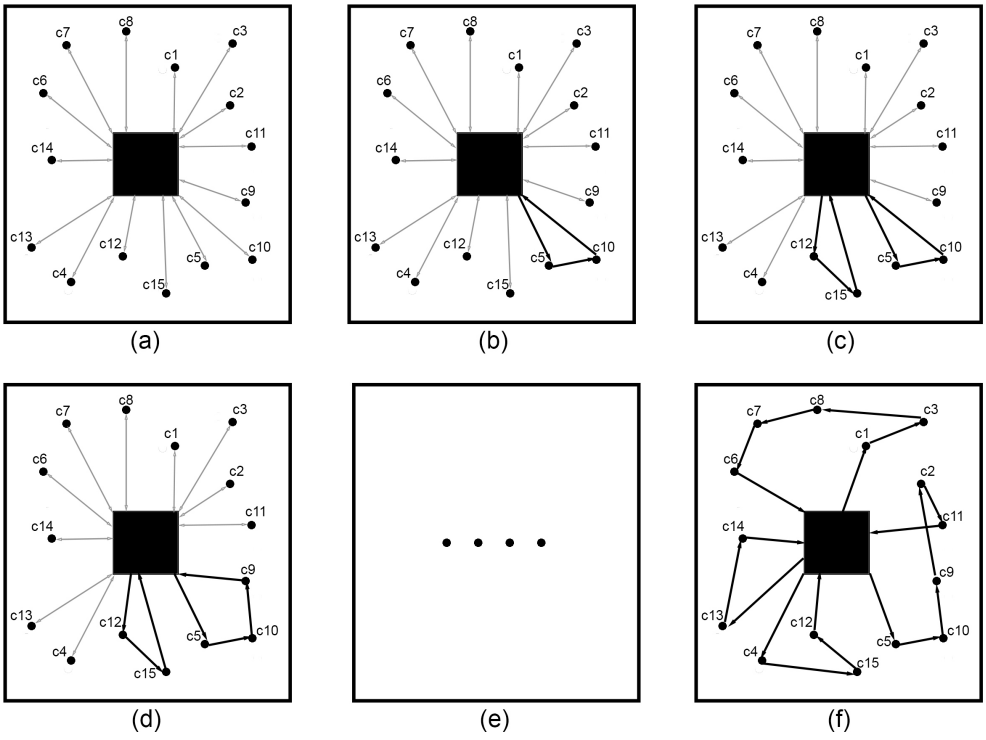


Figure 12: Execution of the savings heuristic on a CVRP.

The term “closed” route is used to denote route where customers can’t be inserted anymore. Insertion heuristics is then divided into sequential insertion and parallel insertion. Sequential insertion heuristics builds one route at a time while parallel insertion heuristics build many or all routes in parallel. Sequential insertion discussed in R. H. Mole 1976 is shown in the figure 13.

Insertion heuristic starts solving a problem from scratch. It constructs a set of routes by deciding which customer to insert at which position in which route at each iteration. The family of insertion heuristics is differentiated on the basis of location and sequence of customer insertion in routes. One insertion heuristic could be to insert the customer that keeps the overall cost as least as possible. Another heuristic could be to keep one route open at a time and choose the nearest feasible customer to be inserted in the open route next to the currently last visited customer in the open route. Some complex version can consider checking the best possible position in the open route by going through the whole route from start to end.

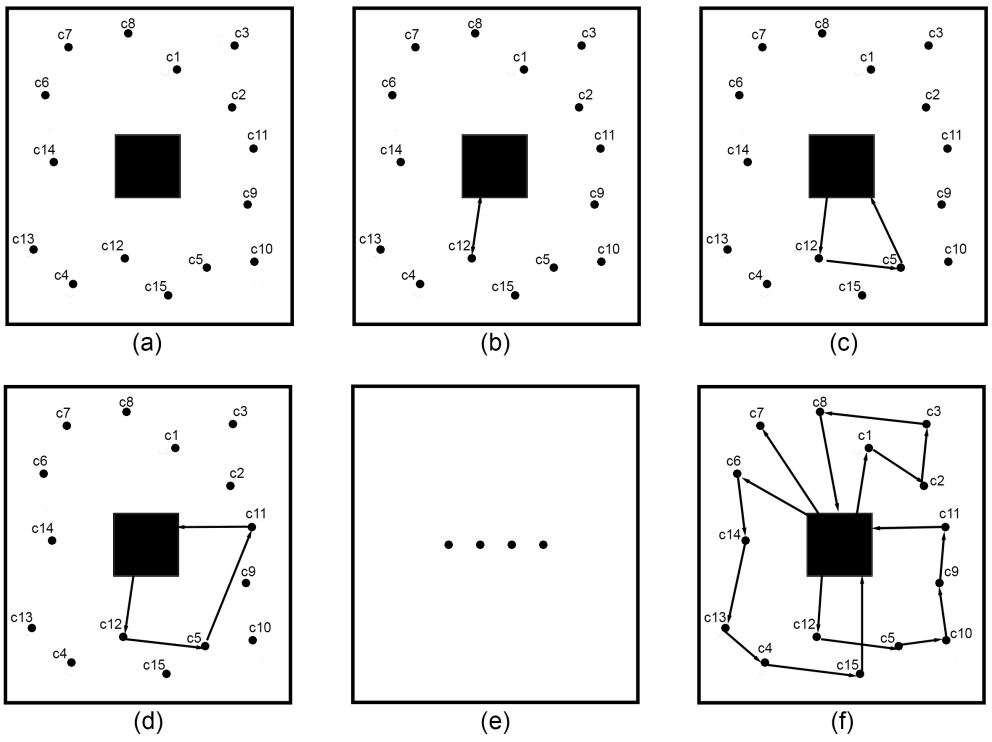


Figure 13: Execution of the Mole and Jameson Heuristic on a CVRP.

In Toth and Vigo 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 routed customer i in route r_{cur} such that $q_i + demand(r_{cur}) \leq Q$ is calculated. This is done in three steps. First, the minimal detour α_{ip} of the visiting the customer in the currently open route is determined. The cost inserting i in every position $p(1 \leq p \leq |r_{cur}.S| + 1)$ in r_{cur} is computed as $\alpha_{ip} = c_{r_{cur[p-1]i}} + c_{ir_{cur[p]}} - \lambda c_{r_{cur[p-1]}r_{cur[p]}}$ where λ is a parameter. The λ_{ip} value is minimal for some p, and this minimized value corresponds to the smallest possible detour. Next, the distance that is saved by visiting i in r_{cur} rather than in a route of its own is evaluated. This is done along the formula $\beta_i = \mu c_{0i} + \alpha_{ip}$ where μ is a parameter. Finally, the customer i maximizing β_i is selected and inserted in the position p of R_{cur} causing the minimal detour ($r_{cur} = insert(i, r, p)$). In the third step, the resulting route is then optimized using a 3-exchange(3-opt) optimization (see 4.7). Once the open route r_{cur}, r_{cur} get saturated and can not accomodate more customers feasibly then it is closed and added to $Sol(Sol = Sol \cup \{r_{cur}\})$. After this, a new empty route r_{cur} is opened and the above procedure is repeated until all customers have been included in a route and the routes have

been added to *Sol*. The number of vehicles is not used as a decision variable in forming the *Sol* so the routes can be more than the available vehicles. The execution of the Mole and Jameson Heuristic is visualized in figure 13.

4.6.3 Sweep Heuristic

The Sweep Heuristic is an instance of a so-called *Cluster first, route second* heuristic. They are also known as clustering algorithms. They are two phase algorithm. The first phase consists of partitioning customers into sets (clusters). The second phase creates one route per cluster, computing the visit to all the customers in that cluster. This latter step is typically implemented by solving a Traveling Salesman Problem per cluster. A third phase may be employed to repair the solution if the route formed in the second phase can not be serviced by a single vehicle.

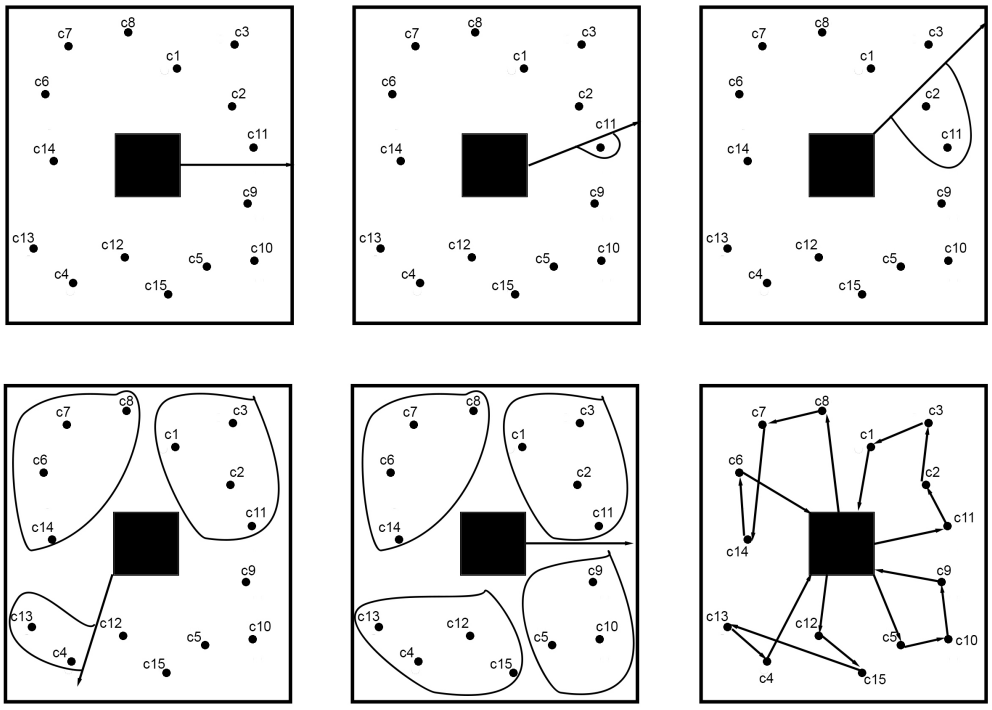


Figure 14: Intermediate steps in the execution of the Sweep Heuristic on a CVRP. Note that the Sweep heuristic is designed for instances where the customers are distributed in clusters.

In Sweep Heuristic, we consider the vertexes in V to be distributed on a plane. With each

customer $i \in V\backslash\{0\}$ are associated its polar coordinates w.r.t. the depot (θ_i, ρ_i), while for some customer $j \in V\backslash\{0\}\theta_j = 0$. The heuristic starts with a empty current route $r_{cur}(r_{cur}.S = \varnothing)$. A ray centered at the depot performs a full circle rotation, sweeping over the customer vertexes, in such a way that a customer $j \in V\backslash\{0\}$ such that $\theta_j = 0$ is encountered first by the ray. The moment customer v is get in the focus of the ray, if $demand(r_{cur}) + q_v \leq Q$ it is added to the set of customers visited in the currently open route $r_{cur}(r_{cur}.S = r_{cur}.S \cup \{v\})$. If the customer v cannot be added to $r_{cur}.S$ then $r_{cur}$is closed and added to $Sol(Sol = Sol \cup \{r_{cur}\})$, a new empty route r_{cur} is opened and v is added to this new route. Once a full ray rotation has been performed the current route r_{cur} is added to Sol and all customers are visited in Sol. Then, for every route r in Sol is treated as a traveling salesman problem to decide the optimal sequence for visiting the vertexes in the set $r.S \cup \{0\}$. Once this sequence has been determined $r.\sigma$ is adapted accordingly. The Sol might contain more routes than available vehicles because vehicles are not a decision variable while forming the Sol. An example of the execution of the Sweep Heuristic on a CVRP instance is given in figure 14.

4.7 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 evaluating the effect of changing the solution in a systematic way. If one of the changes leads to an improved solution, 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. The solution found to be optimal may not be globally optimum. The term improvement heuristic in (Laporte and Semet 2001) is used to describe a local search heuristic that only performs moves that improve the value of objective function of the solution. 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 for finding a much better solution after a few steps in the solution search space. Local search algorithms are useful (Russell and Norvig 2003) for solving pure optimization problems, in which we try to find

the best state according to an objective function .

Local search can be better explained with the help of state-space landscape as shown in the figure 15 from discussion in Russell and Norvig 2003. The location in the landscape represents state of the solution and elevation represents value of heuristic cost function for that state. If elevation is taken as the cost of 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 global optimal solution. A local optimum is a solution that is optimal only in its surrounding neighborhood (i.e. none of the neighboring solutions can improve it) while a global optimum is absolutely optimal through the whole state space landscape.

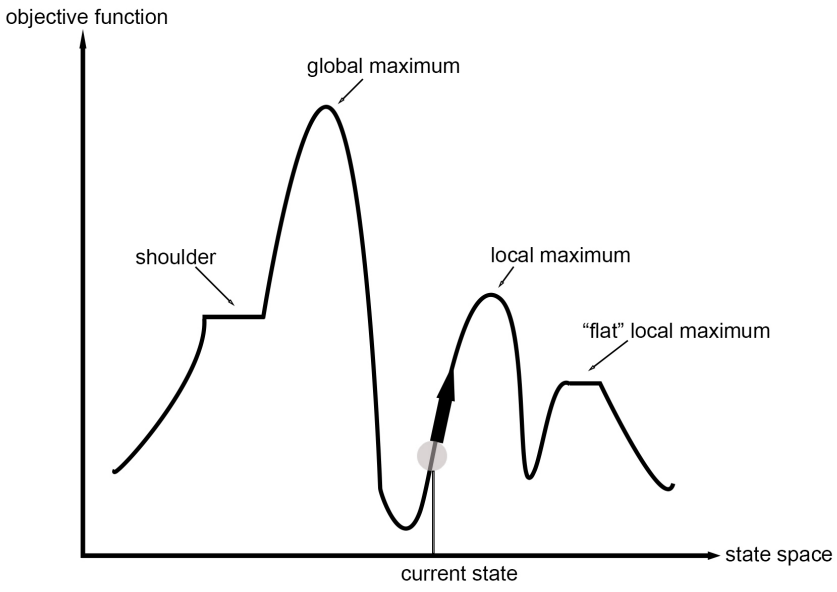


Figure 15: A one-dimensional state-space for local search.

Given a current solution *Sol* the set of solutions that can be obtained by performing an operation op on *Sol* is called the neighborhood $N_{op}(Sol)$ of *Sol*. The operation op is performed

by applying a neighborhood operator $h_{op}(..., Sol)$ to the current solution. Such an operator generally uses several parameters indicating which parts of the current solution will play a role in influencing it. The size of the neighborhood $N_{op}(Sol)$ of Sol depends on the different parameters considered for $h_{op}(..., Sol)$. The neighboring solution of the current solution is constructed and evaluated at each iteration in the local search. One of the neighboring solutions is then selected as the new current solution. Note, that often more than one neighborhood operator is used in local search.

Intensification and *Diversification* are important concepts in Local Search. Intensification means that the search is concentrated in a specific area of the solution state space which seems to be producing promising results (typically with the goal of ending up at a local optimum). Diversification means that the search is randomized to explore different parts of the solution state space in order to make sure different areas are covered and the search does not give the same local optimum for different runs. Intensification and Diversification measures are commonly implemented in the evaluation and selection of neighboring solutions.

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 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 evaluates the neighboring solutions while the construction of the neighborhood is in progress and as soon as a neighboring solution with a better quality than the current solution is found, the search moves to it. 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. 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 Best improvement the complete neighborhood is constructed and evaluated. The solution improving the solution quality the most is selected as new current solution. Best Improvement is also called greedy hill-climbing or discrete gradient descent.

In a randomized evaluation, orderings of the Best improvement strategy out of the ω best neighboring solutions is selected at random (ω being a parameter).

4.7.1 Adaptation to the CVRP

Local Search has been applied to Vehicle Routing Problems on a wide scale and has been proven to be efficient especially on large-scale instances where exact methods are intractable. In this section, examples of how the main steps for local search are handled for CVRP are given.

1. **Construction of the initial solution**

   The initial solution is populated using construction heuristics such as the ones presented in 4.6. In order to generate a number of initial solution for the same set of customers, we can take some meaningful random steps. An example of 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. A feasible solution should be for example build as many routes as the number of vehicles in the fleet, or every route should satisfy the capacity constraints.

2. **Solution evaluation**

   In case of 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 could for example 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 infeasible solution is to make the solution feasible by restructuring the solution, later on if no feasible solution was found directly. It is also possible to direct the search towards feasible solutions by penalizing infeasible solutions (the higher the infeasibility the higher the resulting objective value). Generally the violation of structural constraints such as the number of visits to a customer or routes starting and ending at the depot is not allowed.

3. **Construction of the neighborhood**

There are two major catagories of VRP neighborhoods: Single-Route Improvements and Multiroute 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 Multiroute Improvements customers are exchanged and moved between two or more routes at a time. The impact of Multiroute 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 use an infeasible solution as an initial solution then the search procedure should be allowed to move to infeasible neighboring solution. This decision is taken because it might be impossible to find feasible solution in the neighbourhood of an initial infeasible 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 solution. The decision of constructing a neighborhood which may or may not contain infeasible solutions is implemented by choosing parameters for the corresponding neighborhood operator.

Computing the full neighborhood means considering the change by a neighborhood operator in all possible parts of the current solution. To compute a reduced neighborhood we save execution time by avoiding to evaluate all neighbors. This is done by selecting a subset of part of current solution to be considered for being changed. The construction of full neighborhood needs more computation time as compared to reduced neighborhood but the results are also relatively more improved solutions.

Neighborhood operators makes a vital part of Local Search procedures. To compute the neighborhood of the current solution, a neighborhood operator is selected from a list of operators. Several well-known neighborhood operators for the CVRP are presented below.

(a) ***Relocate operator***

The relocate operator $\eta_{reloc}(i, r_1, p, r_2, Sol)$ $(r_1, r_2 \in Sol)$ takes a customer i currently visited in $r_1 (i \in r_1.S)$, removes it from r_1 and reinserts it at position p in route $r_2 (1 \leq p \leq |r_2.S| + 1)$. The size of $N_{reloc}(Sol)$ is determined by the different routes, customers and positions considered for η_{reloc}.

Example: Let $Sol = \{r_1, r_2, r_3, r_4\}$ with $r_1.\sigma = \langle v_a, v_b, v_c \rangle$ and $r_2.\sigma = \langle v_d, v_e, v_f \rangle$. Then $\eta_{reloc}(v_b, r_1, 2, r_2, Sol)$ results in the modified solution $Sol' = \{r_1', r_2', r_3, r_4\}$ with modified routes $r_1'.\sigma = \langle v_a, v_c \rangle$ and $r_2'.\sigma = \langle v_d, v_b, v_e, v_f \rangle$

(b) ***Swap operator***

The swap operator $\eta_{swap}(i, r_1, j, r_2, Sol)$ $(r_1, r_2 \in Sol)$ takes two customers i and j visited in different routes r_1 and r_2 $(r_1 \neq r_2, i \in r_1.S\,,\ j \in r_2.S)$ and exchanges them. The size of $N_{swap}(Sol)$ is determined by the different routes and customers considered for η_{swap}.

Example: Let $Sol = \{r_1, r_2, r_3, r_4\}$ with $r1.\sigma = \langle v_a, v_b, v_c \rangle$ and $r2.\sigma = \langle v_d, v_e, v_f \rangle$. Then $\eta_{swap}(v_b, r_1, v_d, r_2, Sol)$ results in the modified solution $Sol' = \{r_1', r_2', r_3, r_4\}$ with modified routes $r_1'.\sigma = \langle v_a, v_d, v_c \rangle$ and $r_2'.\sigma = \langle v_b, v_e, v_f \rangle$

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

The k-exchange operator $\eta_{kex}(r_1, A_1, A_2, Sol)$ $(r_1 \in Sol \text{ and } A1 \cap A2 = \varnothing)$ partitions a route r_1 into $k+1$ segments by dropping all arcs in A_1 from $r_1 (A_1 \subseteq \cup_{j=0}^{|r.S|} \{ (r[j], r[j+1]) \})$ and reconnecting the resulting segments using the arcs in A_2 $(A_2 \subseteq \cup_{j=0}^{|r.S+1|} \{ (\delta^-_{r[j]} \cap \cup_{l=0}^{|r.S|+1} \delta^+_{r[l]}) \})$. Note, that some of the route segments may be reversed in the resulting route. The size of $\eta_{swap}(Sol)$ is determined by the different routes and arc sets considered for η_{swap}.

Example: Let $r_1.\sigma = \langle v_a, v_b, v_c, v_d, v_e \rangle, A_1 = \{(v_a, v_b), (v_d, v_e)\}$ and $A_2 = \{(v_a, v_d), (v_b, v_e)\}$. Then $\eta_{2ex}(r_1, A_1, A_2, Sol)$ results in the modified solution $Sol' = \{r_1', r_2, r_3, r_4\}$ with modified route $r_1'.\sigma = \langle v_a, v_d, v_c, v_b, v_e \rangle$.

(d) ***Cross operator***

The cross operator $\eta_{cross}(i, r_1, j, r_2, Sol)$ $(r_1, r_2 \in Sol)$ exchanges the segments starting with i and j and ending at the depot, in routes r_1 and r_2 $(r_1 \neq r_2, i \in r_1.S, j \in r_2.S)$. The size of $N_{cross}(Sol)$ is determined by the different routes and

customers considered for η_{cross}.

Example:Let $r_1.\sigma = \langle v_a, v_b, v_c \rangle$ and $r_2.\sigma = \langle v_d, v_e, v_f \rangle$. Then $\eta_{cross}(v_b, r_1, v_d, r_2, Sol)$ results in the modified solution $Sol^{'}$ with modified routes $r_1^{'}.\sigma = \langle v_a, v_d, v_e, v_f \rangle$ and $r_2^{'}.\sigma = \langle v_b, v_c \rangle$

4.8 Metaheuristics

Metaheuristic is a higher level heuristic used to select another heuristic that may provide a better solution to an optimization problem. A metaheuristic refers to an iterative master strategy that guides and modifies the operations of subordinate heuristics by combining intelligently different concepts for exploring and exploiting the search space. It is useful specially when we have very little or incomplete information or limited computational power to know the global optimum solution to a problem in advance (Luke 2013). They are not problem specific algorithms, it is nonetheless necessary to do some fine-tuning of its intrinsic parameters in order to adapt 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 to get stuck in local optima but it does not guarantee to find the globally optimum solution. The advantage of metahueristic is the use of less computational power as compared to other optimization problem.

Metaheuristic can be broadly catagorised into population based methods and local search methods. Local Search Methods explore the neighborhood of a solution in subsequent iterations. Methods based on local search include simulated annealing (see Gelatt and Vecchi 1983 and Nikolaev and Jacobson 2010), deterministic annealing (see Dueck 1993, Dueck and Scheuer 1990 , and Li, Golden, and Wasil 2005), tabu search (see Glover 1986 and Gendreau and Potvin 2010), iterated local search (see Baxter 1984 and Lourencco, Martin, and Stutzle 2010), and variable neighborhood search (see Mladenovic and Hansen 1997). Population-based heuristics evolve a population of solutions which may be combined together in the hope of generating better ones. This category includes ant colony optimization (see Reimann, Doerner, and Hartl 2004), genetic algorithms (see Holland 1975 and Prins 2004), scatter search, and path relinking (see Glover 1977 and Resende et al. 2010).

4.9 Ant Colony Optimization

Ant Colony Optimization (ACO) has been explained in great detail in 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. Ants generate and emit these hormonal chemicals in order to relay a message to another member of the colony. Ants produce different types pheromones, each with its own purpose. Ants emit pheromones 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 colony with food to attract other ants for following the path to food location. When choosing between several paths the ants tend to take the path with highest pheromone presence. If the ants have to chose from several long and short paths between the colony and food location , the pheromone's presence on the shorter path will increase more quickly because the time need to traverse is less than the time required for longer path. This causes the ants to follow shorter path and deposit even more pheromones, converging the majority of ants to follow the shorter path.

In the above example, ants behave as intelligent agents and try to constructs solutions to the problem being optimized. Ants starts 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 afterwards its evaporation provides the decision making process to modify the current solution forming method. Each ant in the colony continues to work until it has built a full solution or hit a dead end in enhancing the current solution in a feasible way at any point. The solutions are often optimized using a local search. In each iteration of ant colony optimization, an entire colony consisting of l ants is executed. At the end of such an iteration, the current solution is possibly updated and pheromones are deposited on the paths l, that ants took to build their solutions. This is applied in proportional measure to the quality of the given solutions.

Ant Colony Optimization can be used to solve Vehicle Routing Problems due to similarity

of find optimized set of paths in the problem graph. Like vehicles, ants select an arc in the problem graph to add to current solution Sol_{par}. After selecting an arc the ant will deposit pheromones on the arc taken thus making it part of the constructed solution.

4.9.1 Adaptation to the CVRP

Ant Colony Optimization has been used to solve various kind of CVRP variants as discussed in Reimann, Stummer, and Doerner 2002 , Gambardella, Taillard, and Agazzi 1999 and Fuellerer et al. 2010. We discuss basic ant colony optimization where the ants used a simple insertion heuristic. When adapting an Ant Colony Optimization approach to a particular problem the following design choices need to be taken:

- ***Contructing the solution***

  To construct the solution each ant executes the construction heuristic as discussed in 4.6. An ant starts at the depot and then at each step selects the next vertex to move to. Hence an ant can construct one route at a time. At each step it will either choose a destination vertex from unchosen vertexes or move back to the depot. If at any step the ant chooses to move back to 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 number of routes is greater than available vehicles, we can either consider it a failed solution or post optimize the solution to make it feasible. Local search can be used for post optimization.
- ***Selecting the next step***

  The ant chooses a new destination vertex at each step, which is analogous to to choosing an arc to add to its path. The ant choses next arc randomly but its selection is still biased towards arcs with a higher pheromone deposit. Let Sol_{cur} be the current set of closed routes, r_{cur} the current open route and v_{cur} the vertex the ant last added to its path. $Ex(Sol_{cur} \cup r_{cur})$ is the set of vertexes that can be feasibly used to extend r_{cur}. $Ex(Sol_{cur} \cup r_{cur})$ is then defined as $\{j \in V \setminus Vis(Sol_{cur} \cup r_{cur}) \mid demand(r_{cur} + q_j \leq Q)\}$, where $Vis(R) = \cup_{r \in R}\{r.S\}$ is the set of vertexes already visited in the routes in R. The probability associated with visiting vertex j next is then given by

$$p_{v_{cur}j} = \begin{cases} \frac{\tau^{\alpha}_{v_{cur}j}.\eta^{\beta}_{v_{cur}j}}{\sum_{s \in Ex(Sol_{cur} \cup r_{cur})} \tau^{\alpha}_{v_{curs}}.\eta^{\beta}_{v_{curs}}} & if\ j \in Ex(Sol_{cur} \cup r_{cur}) \\ 0 & otherwise \end{cases} \tag{4.1}$$

where $\tau_{v_{cur}j}$corresponds to the pheromone deposit on arc (v_{cur}, j) and $\eta_{v_{cur}j}$to heuristic information. This information can be for example the inverse of the distance associated with arc (v_{cur}, j), i.e. $\tau_{v_{cur}j} = \frac{1}{c_{v_{cur}j}}$. .

- ***Post-optimizing the solution***

  The solution built by an ant is commonly post-optimized using a Local Search approach as seen in 4.7.

- ***Updating the pheromone deposit***

  After finishing one complete iteration of the ant colony optimization, the pheromone deposit on the graph is updated. In order to save the system from converging too quickly the pheromone quantity on every arc in the problem graph is evaporated. Then the solutions produced by all ants during this iteration are used to update the pheromone deposit on the arcs of the problem graph. The pheromone deposit on arc (i, j) is updated according the following formula:
  $\tau_{ij} = p.\tau_{ij} + \sum_{k=1}^{l} \sigma^{k}_{ij}$.

  where $p(0 \leq p \leq 1)$ is called the trail persistence and corresponds to the fraction of the current pheromone quantity τ_{ij} that remains on (i, j); where σ^{k}_{ij} corresponds to the quantity of pheromones deposed by ant k. This latter quantity is 0 if arc (i, j) doesn't appear in the solution constructed by ant k and else depends on to the total cost of the solution.

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 is 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 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 Capacitated Vehicle Routing Problem (CVRP). The CVRP is about delivering service to a set of customers with associated demands using a given set of vehicles. Each vehicle is considered to have 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 does not exceed the threshold limit of the truck. The majority of vehicle routing problem consider the capacity constraint as a threshold of weight 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 world logistics where the shape of the product is considered. The items to be packed are distinctly rectangular which cannot be stacked on top of each other due to fragility, weight or large dimensions. 2LCVRP is useful in real life scenarios like transporting large kitchen appliances such as refrigerators or catering equipment like food trolleys.

The capacitated vehicle routing problem with three-dimensional loading constraints (3LCVRP) is a combination of vehicle routing and three dimensional (width,height,depth) bin packing problem. This problem was introduced by Gendreau et al. 2006b for considering loading of rectangular boxes in a fleet of vehicles while optimizing routes. The objective is to minimize the traveling costs while providing a feasible loading for each vehicle. Each item in 3LCVRP is considered as a cuboid with weight. In general, a cuboid is an object with dimension as width, height and depth. It has six rectangular faces. The vehicle volume capacity is expressed as 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. 2006b 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 5.4 section.

5.3 Related Work

In the three-dimensional capacitated vehicle routing problem, the three dimensions of the vehicle are taken into account and the customer's demand also consists of three dimensional items. This problem was first addressed by Gendreau et al. 2006b. They modeled sequence-based loading, stacking and vertical stability constraints. The vertical orientation of the items

in vehicles was fixed. 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 Toth and Vigo 2001) for a detailed description of CVRP test bed instances). The arc costs are determined as the Euclidean distances between coordinates of customers. The vehicle loading volume has dimensions W = 25, H = 30, and L = 60. For each customer the number of required items is randomly generated according to a uniform distribution between 1 and 3. Each item dimension is randomly generated according to a uniform distribution in the interval between 20% and 60% of the corresponding vehicle dimension. The minimum supporting area is set equal to 0.75.

This paragraph discuss other research articles that solved the problem discussed in Gendreau et al. 2006b and tested their approach on the test bed provided with it along with some new test instances. 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 improvement of 6.43% over Tabu Search Gendreau et al. 2006b. 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 improvement of 3.13% over Tabu Search Gendreau et al. 2006b. 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 Gendreau et al. 2006b and improved in certain instances from previous results of guided tabu search (GLS) Tarantilis, Zachariadis, and Kiranoudis 2009 and ant colony optimization (ACO) Fuellerer et al. 2010 solutions. In Miao et al. 2012 , a hybrid approach is introduced, which combines Genetic Algorithm for vehicle routing problem and Tabu Search for three dimension loading. This hybrid genetic algorithm obtain new best solutions for several instances. Ren, Tian, and Sawaragi 2011 proposed a hierarchical method to solve the problem. In the subordinated module, a branch and bound method is applied to the modified 3L-CVRP in which the loading constraints are relaxed and replaced by a volume-ratio constraint. In the next step, a container loading algo-

rithm 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 test bed.

In this paragraph we discuss research articles published on vehicle routing involving pick up and delivery(VRPPD) with multiple vehicles and loading constraints. 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. It develops three branch-price-and-cut algorithms to solve the problem exactly for instances with maximum 75 requests. Fagerholt et al. 2013 proposes a formulation for the VRPPD with time windows, complete-shipment constraints and connectivity constraints. It uses this approach to solve real-life ship routing and scheduling problem that arises in tramp shipping. Tramp shipping refers to the trade practice of ships and boats which does not have a fixed schedule or published ports of call. They trade on the spot market with no fixed schedule or itinerary. 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 devise a two stage approach for solving the problem. In the first stage, number of vehicles is minimized using simulated annealing and ejection pool. The second stage minimizes the total travel distance using variable neighborhood search and probabilistic tabu search. In Malapert et al. 2008 , a framework is proposed to solve two dimensional VRPPD with multiple vehicles and sequence-based loading. It develops a constraint programming loading model based on a scheduling approach. The article highlights that most packing techniques use reduction procedures which are not compatible 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 modifies a well-known large neighborhood search for the 1D-PDP Ropke and Pisinger 2006. A tree search heuristic is responsible 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 the all vertexes, i.e. pickup and delivery points including the depot locations at 0. Each edge $e_{ij} \in E$ between v_i and v_j has an associated routing cost c_{ij} and the travel costs is symmetric i.e. $c_{ij} = c_{ji}$ $(0 \leq i\,,\ j \leq 2n\,,\ i \neq j)$. We have got n customer requests each consisting of a pickup point i, a delivery point $n+1$ and a set I_i of goods that are to be transported from i to $n+i$ $(i = 1, \ldots, n)$. Set I_i includes m_i cuboid pieces and each I_{ik} has length l_{ik} , the width w_{ik} and the height h_{ik} $(i = 1, \ldots, n, k = 1, \ldots, m_i)$. Every customer has a time window $[a_i, b_i]$ where a_i corresponds to the beginning and b_i to the end of the time window. It is assumed that $a_i \geq 0$ and $b_i > 0$. The depot has a time window $[a_0, b_0]$ and each vehicle must leave the depot at instant $a_0 = 0$ that corresponds to the beginning of the depot time window and arrived before instant $b_0 > 0$ that corresponds to the end of depot time window. A fleet of T different types of vehicles is located at the depot and each type of vehicle $t (t = 1, \ldots, T)$ 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 is a sequence of four of more nodes starting and ending at a depot. There might be a single hub in the route to which items are delivered. A customer should only appear once in the route sequence. The total cost of the route is the sum of all comprising edges. A solution to 3L-HFCVRPTW is a set of v feasible routes such that each customer is visited exactly once and a valid packing plan is provided for the route. 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 package from one vehicle to another.

5.5 The Hybrid Algorithm

In the sequel, we describe a hybrid algorithm for the 3L-HFCVRPTW consisting of two separate procedures for routing and packing. The routing procedure is developed by professor

Olli Braysy of Myopt Consulting Oy. Boxes are loaded into vehicles by using a variation of Largest Area Fit First (Gürbüz et al. 2009).

5.5.1 Routing Algorithm

The routing part of the 3L-HFCVRPTW is addressed by ejection chain based route reduction. It is a kind of construction heuristics see 4.6. A constructive heuristic starts solving a problem from scratch. In the process of constructing routes the goal is to keep the distance of the solution as small as possible in case we are minimizing the total distance. Route construction heuristic work towards solution of the problem by inserting customers one at a time into partial routes until a feasible solution is obtained. Routes that do not contain any customer are deleted from the solution. This heuristic continues merging routes until no more merging is possible. After the initial solution is created, the optimization part 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 feasible.

We always accept any move penalties that results in a better solution than the current one. If we don't find any moves with out penalties, we accept the first move that improves the objective function, or the best move found when none improves over the current solution. The chosen move is executed and recorded in a list which contains all the previously executed moves.

The types of moves between two routes r_0 and r_1 that are considered are:

- Shift move: insert customer $i \in r_0$ at a specific position in r_1 and remove it from r_0.
- Swap move: 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 pick ups.
- Intra-swap move: swap two customers in a route $i, j \in r_0$.

The moves happen in the following order. First, we evaluate all shift moves possible for all pairs of different routes. Then we evaluate all swap moves, trying all possible insertion points. For a particular type of move all routes are iterated in random order. Intra-swap moves are done selectively: we randomly evaluate swap moves. If we are in feasible mode

and already have a feasible neighbor at any point, we skip all subsequent moves that do not lead to a better solution. Construction heuristics gradually builds a feasible solution while tracking the current cost of the solution but they do not contain an improvement phase by itself (Laporte and Semet 2001). For improvement, an ejection chain based local search is employed to choose the best feasible solution amongst all the feasible solutions.

5.5.2 Packing Algorithm

For a given route, a valid packing plan for 3D loading space of vehicle comprises one or more 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 all boxes in a packing plan must not exceed a maximum weight limit of the vehicle.

The 3D packing algorithm used is variation of Largest Area Fit First (LAFF) minimizing height (Gürbüz et al. 2009). It uses a heuristics that places the boxes with the largest surface area first by minimizing the height from the bottom of the container. If shape of an item is not cuboid then the bounding box of that item is considered e.g. cylinders. The width, depth and height of the container is fixed as per specification of the vehicle available. A vehicle can have two containers for packing items i.e. a hauler and trailer.

After determining the values of height, width and depth of the container, the given 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 layer. In this method, the boxes with largest surface area are sorted in descending order. The selected boxes are searched to find a box that has minimum height. Then, box with minimum height is placed in the container while keeping its largest surface parallel to bottom of container. The second placement method allocates space for the remaining boxes to be placed in the same layer while not crossing the height of layer. In this method the boxes with largest volume are placed first in the empty spaces around the box placed in first placement method. When no more boxes can be placed in the current layer using second placement method, a new layer is started using the first placement method.

If we find a feasible 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 coded program is run on a computer with following configuration: Intel Core i5 2.30 GHz 8 GB Ram.

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

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

Type 2 corrections are cube root of volume was calculated for customers whose length, width and height dimensions are missing and upper limit of 2.3 unit was maintained on width and 2.4 unit was maintained on height . The clipped value is adjusted in length =

volume/(width*height)

SECOND DATASET: The Second dataset was created from a file with a set of PDP customer 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 dimension 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 range $[0.40, 13.60]$, width $[0.40, 2.45]$, height $[0.40, 2.65]$ respectively. The volume is calculated as the product of the three dimensions.

5.6.2 Result with respect to time consumption

The processing time (seconds) consumption of integrating the 3D bin packing with vehicle routing pickup and delivery with time windows can be seen in tables 1 and 2. The same experiment was ran three times. The reported time intervals are obtained by taking average over three runs and then rounding to the nearest second. It can be easily seen that the packing algorithm integrated with routing algorithm does not incur a huge time consumption penalty to solving the packing problem together with the routing problem.

First Dataset:

File	***Customers***	***Vehicles***	***Running Time***	***Running Time With Packing***
Type1	138	11	28	53
Type2	138	11	24	26

Table 1: Result of vehicle routing with packing as compared to without packing

Second Dataset: 

File	*Customers*	*Vehicles*	*Running Time*	*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 2: Result of vehicle routing with packing as compared to without packing

5.6.3 Result with respect to route formation

The routing solutions of integrating the 3D bin packing with vehicle routing pickup and delivery with time windows can be seen in tables 3 and 4.The same experiment was ran three times. The reported virtual vehicles and unassigned orders are obtained by taking 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 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. This shows that approximate 3D bin packing gives us more realistic solution which can be implemented on ground saving our time, resources and manual labour.

First Dataset:

File	*Customers*	*Vehicles*	*Virtual Vehicles Packing*	*Virtual Vehicles Without Packing*	*Order Unas-signed Packing*	*Order Unas-signed Without Packing*
Type1	138	11	53	40	88	62
Type2	138	11	60	40	98	62

Table 3: Result of vehicle routing with packing as compared to without packing

Second Dataset:

File	*Customers*	*Vehicles*	*Virtual Vehicles Packing*	*Virtual Vehicles Without Packing*	*Order Unas-signed Packing*	*Order 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 4: Result of vehicle routing with packing as compared to without packing

6 Conclusion

We have integrated a 3D bin packing algorithm which is 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 specially 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 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.

Baker, Brenda S, and Edward G Coffman Jr. 1981. "A tight asymptotic bound for next-fit-decreasing bin-packing". *SIAM Journal on Algebraic Discrete Methods* 2 (2): 147–152.

Baxter, John. 1984. "Depot location: A technique for the avoidance of local optima". *European journal of operational research* 18 (2): 208–214.

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.

Christofides, N., and J. E. Beasley. 1984. “The period routing problem”. *Networks* 14 (2): 237–256. ISSN: 1097-0037. doi:10.1002/net.3230140205. http://dx.doi.org/10.1002/net.3230140205.

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.

Corberán, Ángel, and Gilbert Laporte. 2015. *Arc routing: problems, methods, and applications.* Volume 20. SIAM.

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.

Crainic, Teodor Gabriel, Guido Perboli, and Roberto Tadei. 2008. "Extreme point-based heuristics for three-dimensional bin packing". *Informs Journal on computing* 20 (3): 368–384.

———. 2009. "TS 2 PACK: A two-level tabu search for the three-dimensional bin packing problem". *European Journal of Operational Research* 195 (3): 744–760.

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.

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.

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

Dueck, Gunter. 1993. “New optimization heuristics: the great deluge algorithm and the record-to-record travel”. *Journal of Computational physics* 104 (1): 86–92.

Dueck, Gunter, and Tobias Scheuer. 1990. “Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing”. *Journal of computational physics* 90 (1): 161–175.

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. 2003. *Introduction to evolutionary computing*. 1st edition. 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”.

Fekete, Sándor P, and Jörg Schepers. 1997. “A new exact algorithm for general orthogonal d-dimensional knapsack problems”. In *European Symposium on Algorithms,* 144–156. Springer.

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.

Gambardella, Luca Maria, Éric Taillard, and Giovanni Agazzi. 1999. "MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows".

Gelatt, CD, MP Vecchi, et al. 1983. "Optimization by simulated annealing". *Science* 220 (4598): 671–680.

Gendreau, Michel, Manuel Iori, Gilbert Laporte, and Silvano Martello. 2006a. "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.

———. 2006b. "A tabu search algorithm for a routing and container loading problem". *Transportation Science* 40 (3): 342–350.

Gendreau, Michel, and Jean-Yves Potvin. 2010. "Tabu search". In *Handbook of Metaheuristics,* 41–59. Springer.

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. 1977. "Heuristics for integer programming using surrogate constraints". *Decision Sciences* 8 (1): 156–166.

———. 1986. "Future paths for integer programming and links to artificial intelligence". *Computers & operations research* 13 (5): 533–549.

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".

Holland, John H. 1975. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press.

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

İ. K. Altinel, T. Öncan. 2005. "A New Enhancement of the Clarke and Wright Savings Heuristic for the Capacitated Vehicle Routing Problem". *The Journal of the Operational Research Society* 56 (8): 954–961. ISSN: 01605682, 14769360. http://www.jstor.org/stable/4102067.

Ioannou, George, Manolis Kritikos, G Prastacos, et al. 2001. "A greedy look-ahead heuristic for the vehicle routing problem with time windows". *Journal of the Operational Research Society* 52 (5): 523–537.

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.

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

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.

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

Li, Feiyue, Bruce Golden, and Edward Wasil. 2005. “Very large-scale vehicle routing: new test problems, algorithms, and results”. *Computers & Operations Research* 32 (5): 1165–1179.

Lodi, Andrea, Silvano Martello, and Daniele Vigo. 1999. “Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems”. *INFORMS Journal on Computing* 11 (4): 345–357.

———. 2002. “Heuristic algorithms for the three-dimensional bin packing problem”. *European Journal of Operational Research* 141 (2): 410–420.

Lourencco, Helena R, Olivier C Martin, and Thomas Stutzle. 2010. “Iterated local search: Framework and applications”. In *Handbook of Metaheuristics,* 363–397. Springer.

Lu, Quan, and Maged M. Dessouky. 2005. “A new insertion-based construction heuristic for solving the pickup and delivery problem with hard time windows”. *European Journal of Operational Research* 175:672–687.

Luke, Sean. 2013. *Essentials of Metaheuristics.* Second. 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. 1990. *Knapsack problems: algorithms and computer implementations.* 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.

Mladenovic, Nenad, and Pierre Hansen. 1997. “Variable neighborhood search”. *Computers & Operations Research* 24 (11): 1097–1100.

Motwani, Rajeev, and Prabhakar Raghavan. 2010. “Algorithms and Theory of Computation Handbook”. Chapter Randomized Algorithms, edited by Mikhail J. Atallah and Marina Blanton, 12–12. Chapman & Hall/CRC. ISBN: 978-1-58488-822-2. http://dl.acm.org/citation.cfm?id=1882757.1882769.

Nikolaev, Alexander G, and Sheldon H Jacobson. 2010. “Simulated annealing”. In *Handbook of Metaheuristics,* 1–39. Springer.

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.

Prins, Christian. 2004. “A simple and effective evolutionary algorithm for the vehicle routing problem”. *Computers & Operations Research* 31 (12): 1985–2002.

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, Karl Doerner, and Richard F Hartl. 2004. “D-ants: Savings based ants divide and conquer the vehicle routing problem”. *Computers & Operations Research* 31 (4): 563–591.

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.

Resende, Mauricio GC, Celso C Ribeiro, Fred Glover, and Rafael Marti. 2010. “Scatter search and path-relinking: Fundamentals, advances, and applications”. In *Handbook of metaheuristics,* 87–107. Springer.

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. 2003. *Artificial Intelligence: A Modern Approach.* 2nd edition. 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.

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.

Toth, Paolo, and Daniele Vigo, editors. 2001. *The Vehicle Routing Problem.* 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.

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.

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.