THE UNIVERSITY *of* EDINBURGH

Solution Methods for Some Variants of the Vehicle Routing Problem

Ivona Gjeroska

Doctor of Philosophy
University of Edinburgh
April 18, 2023

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Ivona Gjeroska*)

Acknowledgements

Firstly, I would like to express my deepest appreciation to my supervisor Sergio García for his selfless and continuing support over the years. His guidance, experience and advice have been invaluable. I am also extremely grateful to my second supervisor, Jörg Kalcsics, for his insightful suggestions, advice and support.

I would like to extend my deepest gratitude to the entire research group in OR and Optimization at the School of Mathematics who I have had the honour to get to know over the years. Thank you to Nagi, Minerva, Mook, Tom, Rodrigo, and many others, for their both professional and emotional support over the years, and a special thank you to Barbara and Malte for continuing the legacy of fORum. I cannot begin to express my gratitude to my brilliant friends and colleagues, Paula and Albert, for all of the effort put into the MOPTA competition. I would like to extend my deepest gratitude to Miguel Anjos, as his unparalleled experience, knowledge, guidance and relentless support have been invaluable.

I am also grateful to all of my friends and extended family for their continuing encouragement, support and love.

I am incredibly grateful for my partner, Gavin, for his patience, selflessness, countless cups of coffee brought to my desk and for constantly believing in me.

The most special thank you goes to my parents and my sister for being my biggest cheerleaders, and for literally everything else.

Lay Summary

The package delivery industry faces the problem of finding the most efficient way to visit a group of customers such that everyone receives their orders on time while making the complete operation as fast and cheap as possible. This is *the Vehicle Routing Problem* and its applications spread far past the delivery industry. Some of the applications are: the problem of designing public transportation routes, as well as deciding on the sizes and frequency of vehicles; stock deliveries from warehouses to supermarkets and shops; freight shipping; air transportation, both commercial and freight; building railway networks and roads; waste collection, and many others. All these problems represent a significant fraction of today's economy, and it is of high interest that these costs are minimised.

The Vehicle Routing Problem is one of the classic problems in Operational Research that is quite easy to describe, yet extremely difficult to obtain the best possible solution. Many of the routing problems faced by the industry do not fit the textbook definition of this problem. They contain a variety of features specific to the application in question, making the problem even more complicated. The literature on this problem has a rich collection of solution approaches, some very good in theory but not widely applicable, others theoretically weak yet very efficient in practice. The exploration of both aspects is important as the former helps us understand the problem better, while the latter is crucial for practical applications.

In this thesis we explore two applications of the Vehicle Routing Problem that are particularly good examples of the possible variations and cover a wide variety of added features of the problem. Both problems are inspired by existing companies which have provided us with real data. We explore the challenges of finding the best possible solution and we propose methods that improve the efficiency and accelerate the process. For both problems we propose efficient solution approaches that can easily handle large and complicated instances, and we evaluate their performance.

Abstract

The Vehicle Routing Problem (VRP) is among the most important and widely researched problems in the field of combinatorial optimization. It fits a large variety of applications across many industries faced with the logistics problem of distribution and collection of goods. Depending on the needs of the specific industry, a collection of variants with different properties can be found in the literature. Here we introduce two specific variants of the classical VRP. Then we develop both exact and heuristic solution methods to solve the problems.

Firstly, we introduce a problem motivated by a company that runs a large distribution network in a city in Ecuador. The company operates with a set of trucks and a set of sellers that are routed simultaneously, and a set of retailers that must be visited. The routing is performed in two stages over the course of one week. The sellers visit every retailer exactly once and take their orders. Precisely on the following day, the trucks visit the retailers and deliver the goods. The aim is to find a starting point for the sellers, spread the deliveries evenly across a planning horizon and balance the routes to guarantee fair pay for the employees while minimising the travelling cost.

We propose two mixed-integer models and a branch-and-cut algorithm along with new valid inequalities for routing problems with an unknown depot location. Due to the time and memory limitations of the exact approach, to solve this company's large-scale problem, we developed a heuristic procedure within a tabu search framework. In the construction phase, we introduce an innovative cluster-first route-second heuristic using a combination of proposed alterations of the k-means clustering procedure. A random cheapest insertion routing procedure is applied to each of the clusters. This is an efficient and robust algorithm that can be applied to different VRP variants. For the improvement phase of the heuristic we use short-term memory and intra-day improvements, followed by diversification with inter-day improvements and intensification on the best-known routes. Through extensive computational analysis we prove the efficiency of the outlined solution methods. To the best of our knowledge this problem is new in the literature.

Finally, we present a case study for a capacitated multi-trip VRP with depot location, fleet sizing and time windows. In the first stage, depots are allocated by solving a p-median problem and fleet sizes are determined by identifying the vehicle requirements of several worst-case demand instances. The second stage consists of optimizing the vehicle routes on a daily basis to satisfy the fluctuating customer demand. We assign customers to depots based on distance and routing effort. For the routing problem, we combine a branch-and-cut algorithm and a heuristic with a route construction phase and packing of routes into vehicle trips. This solution approach is a published joint work and the winner of the 12^{th} MOPTA competition.

За моите родители

Contents

Chapter 1

Introduction

1.1 Motivation

In the past few decades a significant amount of work in Operational Research has focused on *transportation problems*. The transportation itself involves all aspects of production and distribution and represents anywhere from 10% to 20% of the final cost of the product [56]. Transportation makes a significant fraction of the economy on a global scale. The GDP from transport in the UK was at £17,339 million in the second quarter of 2022 [124].

With the great development of technology, optimization packages that are based on Operational Research and Mathematical Programming techniques have been widely improved. This has led to substantial savings in global transportation costs, as there are many applications that benefit from an optimized distribution process using computerised procedures. Another important aspect of this success is the improvement made in the development of mathematical models and algorithms. These consider all aspects and characteristics of the real-world applications of different transportation problems, from public transportation route designs, waste collection and stock distribution, to freight logistics and air transportation.

One of the most studied problems of this kind is the transportation of goods from a depot to customers using vehicles. This is known as the *Vehicle Routing Problem* (VRP). Its varied applications and wide range of variants have made it into one of the best-known and most widely applied problems in combinatorial optimization.

1.2 The Vehicle Routing Problem

The VRP was introduced in 1959 when the first article was published by Dantzig and Ramser, called "The Truck Dispatching Problem" [32]. As it is a crucial problem in the field of distribution management, extensive research has been done since its first appearance in the literature. The survey paper by Laporte [77] reviews the most important advances during the first 50 years of research on this problem. So far, many sophisticated exact methods using decomposition algorithms and powerful heuristics have been developed. The wide range of variants of the VRP is owed to the diversity of rules and imposed constraints from real-life applications.

The setup of the VRP involves *customers* that require a certain service and need to

be visited by *vehicles* which are stationed at one or more *depots*. The vehicles travel along a fixed road *network*. The VRP is solved when every vehicle is assigned to an appropriate *route* that satisfies all of the constraints imposed by the problem while minimising the total *cost* of completing the routes. Each vehicle starts and ends at the depot and all of the customers' demand is satisfied.

The VRP is a generalisation of the Travelling Salesman Problem (TSP). Even though the TSP is one of the most famous problems that are difficult to solve, there have been large-scale instances solved to optimality, with a record of 86,000 nodes reported by Applegate et al. [3]. In comparison, in the literature of the classic capacitated VRP, the best exact algorithms can solve instances of up to a few hundred nodes [90, 116]. Most recently, an optimal solution of an instance of 856 nodes and 95 vehicles was reported by Uchoa et al. [126].

In the remaining chapters of this thesis, we will explore routing problems inspired by industry's needs. These problems are defined as a mixture of known *VRP variants*, each exploiting a specific property of the problem. In the following section we will review the variants that are relevant for the later chapters.

1.3 VRP variants

We divide the literature on VRP variants into 4 categories, depending on whether the difference from the classic definition of the VRP is owed to the *vehicle properties*, *customer requirements*, *depot specifications* or the *objective of the problem*.

Vehicle properties

The number and type of vehicles, their capacity, as well as the frequency and amount of time the vehicle can be used, impose important constraints on the problem.

The main property to be mentioned is the *capacity* of the vehicle, and the majority of the VRP variants are related to the *Capacitated VRP* (CVRP). This by itself controls how many customers can be visited in a single route, depending on their demand. Each vehicle can perform only one route, which means that there is a minimum number of vehicles needed to solve the problem. The number of vehicles required to serve the demand of a given subset of customers is calculated by solving a *Bin Packing Problem (BPP)* [93]. A lower bound is often used instead of the BPP, defined as the ratio between the total demand within the subset and the vehicle capacity.

In some cases, the goods delivered are of negligible size. In these problems the customer demand can be set equal to one, and the vehicle capacity can be defined as the maximum number of customers it is willing to serve. These problems are known as *unit-demand VRPs*. This definition is used when route balancing by cardinality is required. Namely, if every vehicle is required to visit a similar number of customers, the unit demand can be used to define a maximum *and minimum* number of customers to be visited by each vehicle on disposal, which imposes an additional constraint on the standard CVRP formulation. This variant is relatively new to the literature, having first appeared during the previous decade [59][60]. As route balancing is the main feature, this problem is called *Balanced VRP* (BVRP) and a polyhedral analysis is given by Bektaş et al. [17]. A different approach is given by

Gouveia et al [58], describing this problem as a VRP with lower and upper bound capacities (LU-VRP) defined through customer unit-demand. The authors introduce reversed multi-star inequalities as part of the problem formulation.

There might be different types of vehicles on disposal, each with different properties. These are *Heterogeneous Fleet VRPs* (HVRP) first introduced by Golden et al. [54].

In some cases, the number of vehicles required for optimal operation is to be decided. These are known as *Fleet Sizing Problems* (FSP). A problem of this type was first introduced by Golden et al. [55] and a significant amount of work has been done on this topic since then. A mathematical formulation for a problem of this type with a heterogeneous fleet is given by Baldacci et al. [10] and a branch-and-price algorithm is described by Pessoa et al. [113]. However, the majority of the solution methods for this problem found in the literature are heuristic.

In the case of the standard VRP each vehicle is assigned to a single route. In some cases, often due to excessive demand or insufficient vehicle capacity, vehicles are allowed to return to the depot, reload, and perform another route. This is often referred to as the *Multi-Trip VRP (MTVRP)* [22].

Customer requirements

Among the most popular and widely researched variants is the *VRP with time windows* (VRPTW). The customers may impose time windows during which they desire to be visited by a vehicle. The VRPTW finds a large range of applications in practice, and has been explored extensively in the literature [115, 74, 92]. These variants often involve a *service time* for each customer.

The customers have a certain *demand* that needs to be satisfied. This refers to the number of goods to be delivered, which is sometimes unknown until the time of delivery. These problems are *Stochastic VRPs* (SVRP), and they are also among the most explored variants in the literature [50, 29].

Often in practice the service that a customer requires is repetitive, and a visit is completed following a recurring schedule. The schedule covers a given planning horizon containing multiple periods, and each customer imposes a rule of frequency for the length of the planning horizon. These are found in the literature as *Period VRP* (PVRP), or *Multi-Period VRPS* (MPVRP). The first appearance of these problems dates back to the 70s [18, 117].

Depot specifications

The properties of the depot – the common point shared by all vehicle routes, define an important group of VRP variants.

A problem may contain multiple depots, and in this case, the problem extends to the assignment of vehicles to depots before the routing stage. This is the *Multi-Depot VRP* (MDVRP). This problem dates back to the 60s when a constructive solution approach was proposed by Tillman [121]. The majority of the research on this problem is based on heuristic procedures, and only a small portion is dedicated to exact methods as shown in a recent survey in [97].

When the location of the depot must be decided simultaneously with the routing, the problem is defined as a *Location Routing Problem* (LRP). This variant has many

applications in practice and it has been widely researched over the past decades, as documented in the survey by Laporte [79]. Recent and improved solution methods can be found in the literature [125, 103].

The vehicles may not be required to return to the depot at the end of the day. In these problems, each vehicle finishes its appointed route at the location of the last customer. This is the *Open VRP* (OVRP), a generalisation of the CVRP. Letchford et al. in 2007 proposed a formulation along with an exact solution method [87].

1.4 Graph theory, definitions and notation

A Vehicle Routing Problem (VRP) has three main components, namely:

- customers and their needs (demand, time windows, etc),
- vehicles with given constraints (limited capacity, working hours) and objectives (to visit all customers obeying their needs) ,
- a travelling cost (often the travelling time or distance between customers).

From the fact that the customers and depot form a network through which the vehicle routes are formed, it is only natural to adopt the terminology of graph theory. The customers can be represented by *nodes* and the shortest paths that a vehicle would travel from one customer to another can be the *edges* in a given *graph*. The *lengths of the edges* are the travelling costs, and with these three components a graph is well defined. In this section we will formally define and discuss some aspects of graph theory that are used in the remaining chapters of this thesis. For further information on the use of graph theory in the context of routing problems we refer the reader to Applegate et al. [3].

A graph $G(N,E)$ is defined as a structure consisting of a finite set of nodes N and a set of edges E. Each edge $e \in E$ joins two nodes $u, v \in N$, said to be the *end-nodes of the edge* e. For the edge $e = (u,v)$, we say that e is *incident* to nodes u and v, and node u is *adjacent* to node v. A real valued function $c : E \longrightarrow \mathbb{R}$ is the *cost function*, appointing a fixed cost to each of the edges in the graph. If for every edge $e = (u,v) \in E$, $c_{uv} = c_{vu}$, the graph is said to be *undirected*, and otherwise *directed*. For this reason, on an undirected graph the order of the end nodes of any edge is irrelevant, therefore $(u,v) = (v,u)$ where $\{u,v\} \subseteq N$. If the graph is directed, $e_i = (u,v)$ and $e_j = (v,u)$ are two different edges.

A graph $G' = (N', E')$ is a *subgraph* of $G(N,E)$ if $N' \subseteq N$ and $E' \subseteq E$. Furthermore, if $E' = \{e \mid e \in E, e \subseteq N'\}$ then G' is a subgraph *induced by* N'. A graph $G(N,E)$ is said to be *complete* if for every pair of nodes $u, v \in N$, the edge $(u,v) \in E$. We define the *degree* of a node v as the number of edges incident to v. A subset $P \subset E$ such that $P = \{(v_1,v_2),(v_2,v_3),...,(v_{k-1},v_k)\}$ is called a *walk*. If $v_i \neq v_j$ for $i \neq j$, P is called a *path* between nodes v_1 and v_k. The length of the walk (path) P is defined as the number of edges $|P|$. Furthermore, if $v_i = v_k$ in a given path P, then P is called a *cycle* or *circuit*. A cycle that contains all nodes in N is called a *Hamiltonian circuit*, or *tour*.

A graph that does not contain any cycles, is called *acyclic*. An acyclic graph is also known as a *forest*. If for every pair of nodes $u, v \in N$ on a given graph G there is a path that connects them, we say that the graph is *connected*, and *disconnected* otherwise. A connected forest that contains all the nodes from N is called a *spanning tree*.

The sets $N_1, ..., N_k \subseteq N$ are *connected components* if they are the maximal sets inducing connected subgraphs of G. A *cut node* is a node $v \in N$ such that the graph induced by $N \backslash \{v\}$ has more connected components than the graph induced by N. In other words, a cut node disconnects the graph. A graph G that does not contain such cut node is called a *2-connected*, or *biconnected* graph. The maximal subgraph that is 2-connected is a *block*.

For a given node set $S \subseteq N$, the set of edges $e = (u, v)$ such that $u \in S$ and $v \in V \backslash S$ is called a *cut* of the graph G. We will use the notation $\delta(S)$ to represent a cut, often called the *co-boundary of* S, $\delta(S) = \{e \in E || e \cap S| = 1\}$. For single node sets, we will use $\delta(v)$ instead of $\delta(\{v\})$. We will denote by $E(S)$ the set of edges for which both end nodes belong to S, $E(S) = \{e \in E || e \cap S| = 2\}$.

1.5 Formulations

In this section we will present three of the most common models for the VRP. These formulations can be extended for the variants mentioned in Section 1.3. The problems are defined on a graph $G(N, E)$, and the depot is located at node 0. Irnich, Toth and Vigo give an extensive analysis of the models available [67]. A detailed overview of the different types of VRP formulations available can be found in Appendix A.

Two-index vehicle flow formulation

This model was introduced for the VRP by Laporte et al. [80, 81]. It is an intuitive formulation using integer variables to represent the number of times a vehicle traverses an edge. As such, it is a direct extension to the most common models for the TSP. When it comes to applying exact methods to the problem, it is often associated with branch-and-cut approaches.

This formulation in its simplest form contains a complicating set of constraints called the *subtour elimination constraints*, often called *capacity constraints*, which control the capacity limits of the vehicles while simultaneously preventing the creation of cycles, or *subtours*. In the case of the CVRP, there is an exponential number of these constraints in the formulation. These constraints can be replaced by a different set of subtour elimination constraints proposed for the TSP by Miller, Tucker and Zemlin [95], known as *MTZ constraints*. They were later extended to fit the VRP formulation [75, 25]. The MTZ constraints result in a compact formulation, which is an advantage over the original flow formulation. However, the linear programming relaxation is significantly weaker [108, 35]. Desrochers and Laporte [36] propose a lifting to strengthen the MTZ constraints for the VRP. Further analysis and improvements are given by Kara, Laporte and Bektaş [70]. However, these inequalities fall outside of the scope of this thesis.

This vehicle flow model is especially useful when it comes to extending solution approaches specialised for the TSP. However, often small changes from different variants are difficult to apply to the model efficiently. For example, when different types of vehicles are on disposal, or when the customer sequence is taken into account in the overall cost of the solution. Moreover, the linear relaxation of this problem can be very weak depending on the additional constraints.

Three-index vehicle flow formulation

The limitations of the two-index vehicle flow formulation can be surpassed by involving additional decision variables. These are variables that identify the vehicle that traverses an edge. The three-index vehicle flow model was first proposed by Golden et al. [57].

As an advantage, this model allows for additional vehicle properties to be included in the model, such as heterogeneity, different work or capacity limits etc. However, especially for instances with a large fleet, the increased number of integer variables and the added symmetry present a significant complication for many solution approaches. For a given vehicle fleet K, each solution has $|K|!$ equivalent solutions obtained by permuting the vehicles. Fischetti et al [43] propose symmetry breaking constraints for this model, however the known solution methods remain intractable [67].

Set partitioning formulation

This formulation proposed by Balinski and Quandt in 1964 [12] uses an exponential number of binary variables, each representing a *feasible route*. The advantage of this formulation is that it only deals with feasible routes, therefore the complicating constraints of any problem variant are not explicitly added to the main model. The objective function is also hidden inside the parameters of the constraints, which allows for complicated (and even non-linear) cost functions to be seamlessly included in the model. The aim is to find an optimal combination of feasible routes, such that all of the customer demand is satisfied. The downside, however, is the large size of the set of feasible routes, that exponentially grows with the size of the problem. This prohibits the direct solution of the problem and demands use of relaxations or *column generation*.

1.6 Exact methods

In the past decades extensive research has been done on exact methods for the VRP, as witnessed by a variety of survey publications [29, 78, 83, 122, 123]. The VRP literature on exact solution methods is mostly divided into two state-of-the-art approaches: branch-and-cut using the vehicle flow formulations, and column generation for the set partitioning models. The remaining chapters of this thesis focus on the former approach. A variety of applications of column generation algorithms for the VRP, including branch-and-price and branch-and-cut-and-price, can be found in the literature [1, 64, 34].

The branch-and-cut algorithm is an extension to the branch-and-bound algorithm where, during the exploration of the branching tree, some valid inequalities are added in the form of *cutting planes (cuts)* with the purpose of tightening the feasible region [99, 118].

Often the problem solved is a relaxation of the original problem obtained by removing complicating constraints. Each candidate feasible solution is then checked against this set of constraints, and only the inequalities that are violated are added back to the model in the form of cuts, making the candidate solution infeasible. The

process of identification of such violated inequalities is called a *separation procedure*. A variety of valid inequalities incorporated in the form of cutting planes have been proposed for the VRP, and many separation procedures for their identification can be found in the literature. In the remainder of this section, we will focus only on several of these that have been implemented in the next chapters.

Capacity inequalities

Due to the exponential number of subtour elimination inequalities (capacity inequalities) in the CVRP formulation, they are often added to the model as cutting planes during the branch-and-cut algorithm. The most common separation procedure in the literature makes use of *connected components* [116, 8]. It is based on the idea of edge shrinking and the creation of supernodes, proposed for the TSP [86]. This idea is often applied for the purpose of identification of other types of inequalities as well [107, 102], since the resulting graph is smaller, easier to handle, and preserves the connectivity of the original graph. When the VRP solution is feasible, this procedure is exact [91].

Lifted subtour inequalities

A strengthened version of the subtour elimination constraints for the asymmetric TSP has been proven to be valid for the VRPTW. When there is no lower bound on the number of vehicles, a *lifting procedure* is performed to raise the left-hand side coefficients of the subtour elimination constraints [63, 44]. The resulting lifted subtour elimination inequalities were successfully incorporated into a branch-and-cut algorithm for the VRPTW [13], separated using shrinking procedures.

Comb inequalities

Originally introduced by Chvátal [26] and formalised by Grötschel and Padberg [62], the Comb Inequalities are among the most widely used valid inequalities for the TSP, right behind the Subtour Elimination Constraints as introduced by Dantzig, Fulkerson and Johnson 25 years prior [31].

Given a complete graph $G(N,E)$, a *comb* is defined as a family of an even number of subsets of N out of which one is a *handle*, and the rest are the comb's *teeth*. Each tooth needs to be connected to the handle, but the teeth cannot be connected to each other. Mathematically, the comb inequality can be formulated as:

$$x(\delta(H)) + \sum_{j=1}^{t} x(\delta(T_j) \geq 3t+1, \tag{1.1}$$

where $H, T_1, ..., T_s \subset N$, and

$$|H \cap T_i| \geq 1, \qquad i = 1, ..., t, \tag{1.2}$$

$$|T_i \setminus H| \geq 1, \qquad i = 1, ..., t, \tag{1.3}$$

$$|T_i \cap T_j| = 0, \qquad 1 \leq i \leq j \leq t, \tag{1.4}$$

$$t = 2n+1, \qquad n \in \mathbf{N}. \tag{1.5}$$

The comb inequalities fit perfectly in the context of routing problems as they do not permit subtour formations. They are a generalisation of the subtour elimination constraints, as they can be obtained from (2.82) by setting $t = 1$ and $|H| = 1$.

The validity of comb inequalities for the TSP was proven by Naddef and Pochet [98], and later generalised for the VRP by Laporte and Nobert [82]. Since then, they have been widely applied in branch-and-cut algorithms [8, 91, 4, 6, 19]. The strengthened comb inequalities, proposed by Lysgaard, Letchford and Eglese in 2004 [91] are shown to generalise and dominate the standard comb inequalities. They take the form

$$x(\delta(H)) + \sum_{j=1}^{t} x(\delta(T_j) \geq S(H, T_1, \ldots, T_t) + 1, \tag{1.6}$$

where

$$S(H, T_1, \ldots, T_t) = \sum_{j=1}^{t} (r(T_j \cap H) + r(T_j \setminus H) + r(T_j)).$$

The majority of the separation procedures for combs implemented within a branch-and-cut framework are inspired from TSP applications. Due to their difficulty, the separation procedures are most often heuristic. Among the more common procedures we will mention the odd-component heuristic [106, 61], the max-back heuristic [101, 102], and the block heuristic [107].

Two-matching inequalities

The two-matching inequalities (also known as blossom inequalities) are simplified combs introduced by Edmonds [41]. Here, each tooth of the comb has exactly two nodes – one on the handle and one outside. Therefore, each tooth T can be identified by an edge on the graph, $T \subseteq \delta(H)$ where H is the handle. Similar separation procedures are used to identify two-matching inequalities and combs. However, for the two-matching inequalities there exists an efficient exact procedure proposed by Padberg et al. [109].

1.7 Heuristic methods

Heuristic approaches for solving the VRP were proposed when the problem was formally introduced in 1959 [32]. Countless efforts have been published since then, involving construction heuristics, solution improvement methods, and ultimately powerful metaheuristics that can generate very good solutions for large scale problems within seconds. Several extensive surveys can be found in the literature [85, 49, 84].

Construction heuristics

Construction heuristics are procedures that create VRP routes from scratch. Creating a VRP solution consists of *clustering* customers into groups assigned to vehicles and *routing* within each cluster. Depending on the order in which these steps are performed, these heuristics can be divided into three categories:

1. **Simultaneously cluster and route.** Among the most popular algorithms in this category is the Clarke and Wright *savings algorithm* [27] based on growing routes through inserting nodes that generate biggest savings in cost. This procedure is widely applied, improved and modified [104, 110, 129, 46]. The very first construction heuristic proposed by Dantzig and Ramser [32] is based on the idea of growing routes through *matching*. It was later implemented on larger instances and further enhanced [37, 127]. The *sequential insertion* heuristic is similar to the savings algorithm [96, 24].

2. **Cluster–first route–second.** The most intuitive approach of this kind is the *sweeping algorithm* [52, 128]. The idea is to take the depot as a centre of a circle that contains all customers within, and sweep the radius around in a given direction to define clusters where the total demand does not exceed the vehicle capacity. The *generalised assignment algorithm* [45] allocates customers to vehicles by solving an assignment problem. For this as well as for the *location-based algorithm*, the number of vehicles is a parameter of the algorithm. The latter solves a location problem to identify customers that can act as centres of their respective clusters. The rest of the customers are then allocated to a cluster by proximity to a centre [21]. Finally, the *petal algorithm* [11] acts similarly to the set covering and column generation principles, by first identifying feasible routes and then joining them together.

3. **Route–first cluster–second.** For this approach a large TSP is solved in the first step before the cycle is split to create routes [16].

Improvement heuristics

These are procedures applied to a previously constructed solution. We split them into 2 categories:

1. **Intra-route improvements.** These are procedures where changes are made within a route. The most widely implemented and well performing procedure is λ*-opt* introduced by Lin [88]. It is based on edge crossing, and the 2-opt and 3-opt heuristics are widely implemented within many complex solution frameworks for routing problems. One of the best performing heuristics for the TSP, the Lin–Kernighan heuristic [89], is based on the same idea. A *generalised insertion procedure (GENI)* was proposed by Gendreau, Hertz and Laporte in 1993 [48] for the TSP. It has since then been widely applied as an improvement heuristic for the VRP and incorporated into many complex metaheuristics.

2. **Inter-route improvements.** In these cases nodes or edges are swapped between two separate routes. Several different implementations can be found in the literature [120, 72].

Metaheuristics

Metaheuristics are robust solution methods that can be applied to a variety of problems. They are based on ideas like neighbourhood exploration, memory, or mimicking natural processes – techniques that can easily be generalised and re-interpreted.

They are among the most widely explored topics for optimization problems today. Several surveys can be found in the literature [84, 51].

1. **Neighbourhood exploration.** These are procedures that exploit the properties of the neighbourhood of a given solution. Some of the most popular algorithms are *iterated local search* [15] and *variable neighbourhood search* [76].

2. **Memory.** One of the best performing metaheuristics for a variety of VRPs is *tabu search*, proposed by Glover in 1986 [53]. This procedure is still one of the best performing VRP procedures [47, 130, 119]. The idea is to keep a record of a number of the most recent changes performed during the course of the algorithm. As long as a move is in the memory, the reverse move cannot be performed. The memory is updated every time a new move is identified. This process prohibits local optima. Additional properties are used to enhance the algorithm, like intensification and diversification. A good overview of this algorithm is given by Cordeau [28]. In this category we can also mention the more recent, *machine learning* based algorithms using neural networks [49].

3. **Natural processes.** Among many others, the most commonly used heuristics for the VRP in this category are *simulated annealing* [73], mimicking the process of heating a metal and then slowly cooling to minimise damage, and the *genetic algorithm* [66, 114], mimicking the process of generation renewal, creating offspring and mutations.

1.8 Thesis outline

The content of this thesis is organised as follows. In Chapter 2 we introduce a VRP variant motivated by the distribution operation of a company with a large customer base. We define this problem as a Two-Stage Multi-Period VRP with Depot Location and Route Balancing. We propose two mixed-integer linear models for the complete problem, accompanied by a branch-and-cut algorithm with some new valid inequalities and corresponding separation procedures. In Chapter 3 we present a heuristic solution method for the same problem. We propose a new construction heuristic procedure with three extensions to a machine learning clustering algorithm. We then present some improvement methods integrated within a tabu-search framework. An appropriate algorithm is used to solve the complete large instance provided by the company in question.

In Chapter 4 we present a published case study. The topic is another variation of the VRP, combining uncertainties, depot location, fleet sizing, time windows and multiple trip allowance. Throughout 10 Sections we present some data analysis and methods for dealing with uncertainties, as well as strategies for depot location and fleet sizing. We propose a mixed-integer linear model for this problem as well as a branch-and-cut algorithm. We design a robust heuristic for the routing stage of the problem. At the end, we present a complete solution of the original problem and performance analysis.

In Chapter 5 we summarise the work presented highlighting the contributions, and we provide a list of ideas and directions for further work.

Chapter 2

A two-stage multi-period vehicle routing problem with depot location and workload balance: Exact methods

2.1 Introduction

The problem studied here is motivated by the logistics aspect of a distribution company based in Ecuador. The company runs a large network of retailers (in this text referred to as *customers*), and each one is visited once per week. On the map in Figure 2.1 the customer locations are shown with black circles, and the depot is the red square. The company has a *next-day delivery* policy, operating with a fixed number of *sellers* who take orders, and *trucks* that deliver the products. To illustrate this, assume that a given customer is visited by a seller on Tuesday and places their order. Then on Wednesday, they are visited by a truck that in turn delivers the order. The company owns one *depot*.

Figure 2.1: Customer map

At the start of each day, all sellers meet their manager at a *certain meeting point* that can vary from day to day, and from there they begin their respective routes. As a final stop they return their lists of orders to the depot, and this is the end of their working day. On the following day, the company sends trucks to deliver the orders from the previous day. The trucks start and finish their routes at the depot. The objective is to minimise the total travelled distance by the trucks and the sellers.

The product delivered is of negligible size so the load capacity and demand quantity can be ignored. Each employee visits roughly the same number of customers on a given day to ensure fair commission-based pay within the company.

The aim is to find a solution that gives *balanced routes* for all sellers and trucks, determines the *best starting point* for the sellers for every day, guarantees *next-day delivery* to all customers, while *minimising the overall travelling cost.*

2.2 Problem definition

The problem can be formally defined as follows: let $G(N,E)$ be a connected, undirected graph, where N is the set of nodes that represents the customers and the depot, and E is the set of edges that represents the shortest paths between customers. Let node 0 be the depot and let $N^0 = N \setminus \{0\}$ be the customer set.

There is a pre-defined *rolling planning horizon* of n^D working days. By 'rolling' we mean that if day $d = n^D$ is the last day of the planning horizon, then we define $d+1 = 1$, the first day of the planning horizon. We have a fixed number of sellers n^S and a fixed number of trucks n^T. As the starting point for the sellers is unknown, an optimal and common starting point $i \in N$ is assigned to the sellers' routes every day, while the ending point is the fixed depot $0 \in N$.

We can divide this problem into two stages for each day d of the planning horizon:

1. The problem of locating a mutual starting point for all sellers and constructing their routes for day d, all ending at the fixed depot, and

2. the problem of constructing the trucks' routes for day $d+1$ over the set of customers defined by the sellers' routes from stage 1.

The next-day delivery rule creates an unbreakable connection between the two problems, and it is the reason they cannot be solved independently while guaranteeing optimality. Therefore, our problem is a *two-stage problem*, where the first stage is the daily allocation of customers, balanced routing, and depot location for the sellers, while the second stage is the construction of balanced routes for the trucks on the defined customer subset. During the length of the planning horizon (working week), every customer must be visited once by a seller and once by a truck. The aim is to describe both stages in one compact formulation.

The remainder of this chapter is organised as follows. In Section 2.3 we focus on the location of the starting point. The main problem is divided into two subproblems introduced in Sections 2.4 and 2.5. They are special cases focusing on only the sellers' and only the trucks' operation, respectively. For each of them we propose two vehicle-flow models and outline their advantages. Finally, in Section 2.6 they are combined to form a two-stage multi-period problem.

2.3 Depot location

The problem itself does not specify the set of potential locations of starting points for the sellers. Here we will show that, under some reasonable conditions, the optimal starting point is at one of the customer locations.

Proposition 2.3.1. *Let $G(N,E)$ be a complete, undirected graph with Euclidean distances where N is the set of nodes and E is the set of edges. Assume that, arbitrarily, nodes $1,..,n^S \in N$ are the first nodes to be visited by each of the n^S sellers after leaving from the starting point x. Assume that the triangle inequality holds for the distances and the starting point is allowed to be anywhere along the edges of the graph. When all of the above conditions hold, there is an optimal solution for which the starting point is at one of the nodes.*

Proof. For any two nodes on the graph $i, j \in N$, we denote by $d(i,j)$ the Euclidean distance between i and j. As we are only concerned with the *starting point* of the sellers, we will consider the subgraph $G^s(N^s,E^s)$ where $N^s = \{1,\ldots,n^S\}$ and $E^s = \{(i,j) \mid i,j \in N^s\}$. Without loss of generality, let us assume that the starting point x is somewhere along edge $(1,2)$, and that $x \neq 1$, $x \neq 2$. Following the assumption that G^s is a complete graph, we will split the set $N^s \backslash \{1,2\}$ into two disjoint subsets $N_1^s, N_2^s \subset N^s \backslash \{1,2\}$ such that:

- $N_1^s = \{i \in N^s \mid d(x,i) = d(x,1) + d(1,i) \leq d(x,2) + d(2,i)\}$ is the set of nodes i for which node 1 is on the shortest path from x to i.
- $N_2^s = \{i \in N^s \mid d(x,i) = d(x,2) + d(2,i) < d(x,1) + d(1,i)\}$ is the set of nodes i for which node 2 is on the shortest path from x to i.

  We want to prove that

$$\exists k \in \{1,2\} \quad \text{such that} \quad \sum_{i \in N^s} d(x,i) \geq \sum_{i \in N^s} d(k,i). \tag{2.1}$$

  Consider the following:

1. For any two nodes $m, n \in N^s$ such that $m \in N_1^s$ and $n \in N_2^s$,

$$\begin{aligned} &d(x,m) + d(x,n) = \\ &d(x,1) + d(1,m) + d(x,2) + d(2,n) = \\ &d(1,2) + d(1,m) + d(2,n) \geq \begin{cases} d(1,m) + d(1,n) \\ d(2,m) + d(2,n) \end{cases} . \end{aligned} \tag{2.2}$$

   The latter holds due to the triangle inequality

$$\begin{aligned} d(1,2) + d(1,m) \geq d(2,m), \\ d(1,2) + d(2,n) \geq d(1,n), \end{aligned}$$

   the symmetric property of the undirected graph, and the fact that x is on edge $(1,2)$:

$$d(x,1) + d(x,2) = d(1,x) + d(x,2) = d(1,2).$$

2. For any node $m \in N_k^s$, $k \in \{1,2\}$,

$$d(x,m) = d(x,k) + d(k,m) > d(k,m). \tag{2.3}$$

The latter holds because $x \neq 1$ and $x \neq 2$ implies that $d(x,k) > 0$ for $k \in \{1,2\}$.

As $N^s = N_1^s \cup N_2^s$ and $N_1^s \cap N_2^s = \emptyset$, we can decompose the left-hand side of (2.1) into a sum of pairs such that, if $|N_k^s| \leq |N_t^s|$ for $k,t \in \{1,2\}$ and $k \neq t$, then for every $i \in N_k^s$ there is a unique $j \in N_t^s$. Let us call this set $N_{pair}^s = \{\{i,j\} \mid i \in N_k^s, j \in N_t^s\}$. The sum $d(x,i) + d(x,j)$ for all of these couples $\{i,j\} \in N_{pair}^s$ satisfies inequality (2.2). Let us define the set $N_{single}^s = N_t^s \setminus \left\{ j \mid \{i,j\} \in N_{pair}^s \right\}$ of the nodes $l \in N_t^s$ that have not been paired yet. Since all of them belong to the same set N_t^s, then they satisfy inequality (2.3):

$$\sum_{i \in N^s} d(x,i) =$$
$$\sum_{\{i,j\} \in N_{pair}^s} \big(d(x,i) + d(x,j)\big) + \sum_{l \in N_{single}^s} d(x,l) \geq$$
$$\sum_{\{i,j\} \in N_{pair}^s} \big(d(t,i) + d(t,j)\big) + \sum_{l \in N_{single}^s} d(x,l) >$$
$$\sum_{\{i,j\} \in N_{pair}^s} \big(d(t,i) + d(t,j)\big) + \sum_{l \in N_{single}^s} d(t,l) =$$
$$\sum_{i \in N^s} d(t,i).$$

This concludes the proof for (2.1), which states that there is an optimal starting point at one of the nodes of the graph. □

Hakimi [65] illustrates a similar conclusion while exploring the problem of locating switching centres in a communications network. In that case, weights are assigned to the nodes as well as the edges, and following a similar logic it is determined that the optimal location of a switching centre in the network is always in a node.

2.4 The seller problem

The first stage of the problem at hand is the sellers' operation. To analyse the features of this problem we study the operation during a single day, removing the multi-period property. In this section we introduce a model for a *VRP with depot location and workload balance*, which following our motivation we will call *The Seller Problem*. As a generalisation of a capacitated VRP, this problem belongs to the class of *NP* problems [111]. For the sake of completeness, we will introduce this problem from scratch.

We define a routing problem for a set of sellers, a set of customers, and a fixed depot. The sellers share a common starting point that is unknown and is part of the decision process, while the ending point is fixed at the depot.

The product distributed is of negligible size, so the routing capacity does not pose a restriction like in the classic CVRP. However, in order to obtain workload balance between the sellers, we impose limits to the *customer set cardinality* within their routes. In other words, each seller visits (roughly) the same number of customers. In practice this is the case with commission workers.

When it comes to modelling, one of the challenges is imposed by the exponential number of possibilities for feasible routes. Depending on the model used (Section 1.5), this challenge appears either as an exponential number of variables (set-partitioning models), or an exponential number of constraints (flow models). We propose two different vehicle flow based models:

1. **Three-index flow model**: three-index formulation with two additional sets of binary decision variables regarding the starting point. This model makes use of the MTZ subtour elimination constraints.

2. **Two-index flow model**: two-index formulation with one additional set of binary decision variables regarding the starting point. For this model we introduce a generalised set of balancing constraints.

The following models are defined on an undirected graph. However, they easily translate to a directed problem.

2.4.1 Three-index flow model

We consider a three-index vehicle flow model where the starting point location is unknown and the routes are balanced. Our claim that the optimal location for a starting point is at one of the customers' locations is proven in Section 2.3. We define the following:

Sets:

N^0: set of customer nodes,

N: $N^0 \cup \{0\}$, set of all nodes.

Parameters:

c^S_{ij}: cost for a seller to travel from node i to node j,

n^C: number of customers,

n^S: number of sellers,

Q^S_{max}: route capacity (maximum route length) for the sellers,

Q^S_{min}: minimum route length for the sellers.

Variables:

$$u^k_{ij} = \begin{cases} 1, & \text{if seller } k \text{ traverses edge } (i,j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

$$z_i = \begin{cases} 1, & \text{if } i \in N \text{ is the starting point,} \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{ij} = \begin{cases} 1, & \text{if } i \text{ is the common starting point,} \\ & \text{and } j \text{ is the first customer visited by some seller,} \\ 0, & \text{otherwise.} \end{cases}$$

σ_i : auxiliary continuous variable for $i \in N$.

The cost matrix is defined by the cost function $c^S : E \to \mathbb{R}_0^+$. In addition we impose a change to the cost function used by Letchford et al. [87]:

$$c_{0i} = 0 \quad \forall i \in N.$$

The u-variables, like the x variables in the classic CVRP, will indicate the complete routes starting and ending at the depot, node 0, and the first edge in each route is traversed at no cost. This allows for the first customer visited by each seller to be arbitrarily far from the depot. Then the new variables y and z are used to determine the *real* starting point. The complete formulation is given below:

$$\min \sum_{i,j \in N} c_{ij}^S \left(\sum_{k=1}^{n^S} u_{ij}^k + y_{ij} \right)$$

$$\text{s.t.} \sum_{i \in N} \sum_{k=1}^{n^S} u_{ij}^k = 1 \qquad \forall j \in N^0, \tag{2.4}$$

$$\sum_{i \in N} \sum_{k=1}^{n^S} u_{ji}^k = 1 \qquad \forall j \in N^0, \tag{2.5}$$

$$\sum_{i \in N} z_i = 1, \tag{2.6}$$

$$\sum_{i \in N} \sum_{k=1}^{n^S} u_{i0}^k = n^S, \tag{2.7}$$

$$\sum_{j \in N^0} \sum_{k=1}^{n^S} u_{0j}^k = n^S, \tag{2.8}$$

$$\sum_{i \in N} y_{ij} = \sum_{k=1}^{n^S} u_{0j}^k \qquad \forall j \in N, \tag{2.9}$$

$$\sum_{j \in N} y_{ij} = n^S z_i \qquad \forall i \in N, \tag{2.10}$$

$$\sum_{i \in N} \sum_{j \in N} u_{ij}^k \geq Q_{min}^S \qquad \forall k = 1, \ldots, n^S, \tag{2.11}$$

$$\sigma_j - \sigma_i + Q_{max}^S \sum_{k=1}^{n^S} u_{ij}^k \leq Q_{max}^S - 1 \qquad \forall i, j \in N^0, i \neq j, \tag{2.12}$$

$$u_{ij}^k, y_{ij}, z_i \in \{0, 1\} \qquad \forall i, j \in N, \quad k \in \{1 \ldots n^S\}, \tag{2.13}$$

$$1 \leq \sigma_i \leq Q_{max}^S \qquad \forall i \in N. \tag{2.14}$$

The objective function minimises the total travelling cost for all sellers. Constraints (2.4)-(2.5) are the 3-index degree constraints. Constraint (2.6) implies that there is only one starting point for all sellers. Exactly n^S sellers should enter and leave the depot, as defined by (2.7) and (2.8). Constraint (2.9) checks whether a given node is the second node visited on a route as defined by the variables u. If so, then constraint (2.10) ensures that it is connected to the new starting point. Constraint (2.11) imposes a lower bound to the size of each of the seller routes. Constraints (2.12) and (2.14) are the MTZ constraints (Section 1.5), and (2.13) impose the binary restrictions.

The route construction process using the formulation above is given in Figure 2.2. The set of nodes N^0 representing the customers is given in Figure 2.2a, and the depot is shown as a square. The seller routes constructed by the u variables are shown as full arrows (Figure 2.2b), and the edges $(0, i)$ are dashed lines as they have non-negative weights in terms of u, but they are zero in terms of c. Each closed route represents a different seller. Figure 2.2c includes the roles of the y and z variables. The edges (i, j) for which $y_{ij} = 1$ are shown with bold lines. These edges connect the starting point to the first customer in a route. In this example $z_6 = 1$, so the starting point is located at node 6. The routes taken into account in the value of the objective function are given in Figure 2.2d. The optimal starting point is at node 6, and the routes of the sellers are $6 \to 7 \to 9 \to 1 \to 0$, $6 \to 5 \to 2 \to 0$, and $6 \to 8 \to 4 \to 3 \to 0$.

In the following we will discuss and clarify some important ingredients in this formulation. These are the main differences with regard to the common VRP formulations.

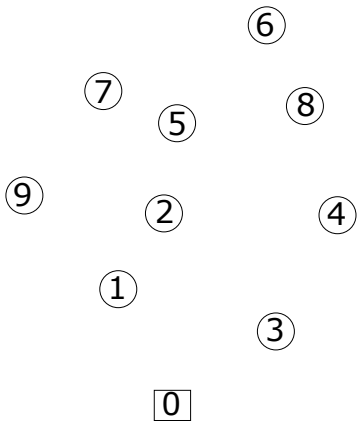
1. **The y variables.** It is desirable to stick to models that have already been explored in the literature, as solution methods fitted to those models can be applied directly. For this reason, we are allowing the u variables to create *closed routes* – this is mended by the y and z variables in line with the problem definition. The z variable is an indicator of the *starting point*, while the y variable connects this starting point to the second node of every route. The remainder of the route is defined by the u variables.

2. **Workload balance.** A problem where it is desirable that all vehicles visit the same number of customers, like in our case, calls for *workload balancing by cardinality*. Similar ideas for balanced workload can be found in the literature, but they most commonly refer to balance by cost/distance, rather than customer set cardinality. This is often a problem with commission workers and a balanced solution would guarantee fair pay. We impose *artificial capacity* to our sellers equal to the number of customers they should visit. A similar idea for workload balancing by cardinality as opposed to distance was proposed by Bektaş et al. [17]. We define the *maximum route capacity* Q^S_{max} for each of the n^S sellers as

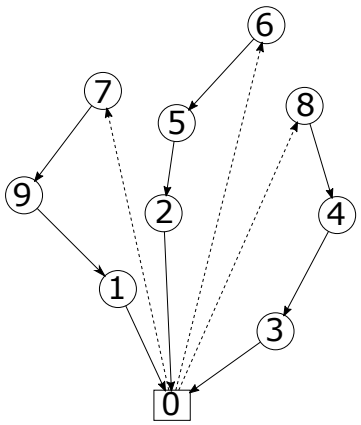
$$Q^S_{max} = \left\lceil \frac{n^C}{n^S} \right\rceil. \tag{2.15}$$

Additionally, we define a *minimum route capacity* for each of the sellers as

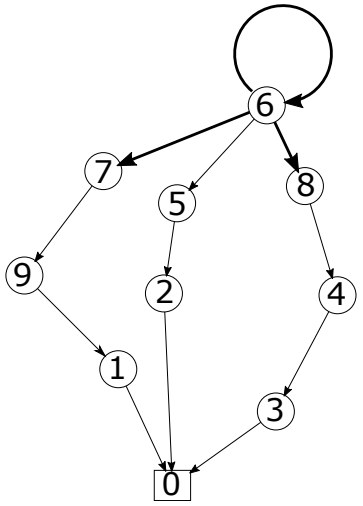
$$Q^S_{min} = \left\lfloor \frac{n^C}{n^S} \right\rfloor. \tag{2.16}$$



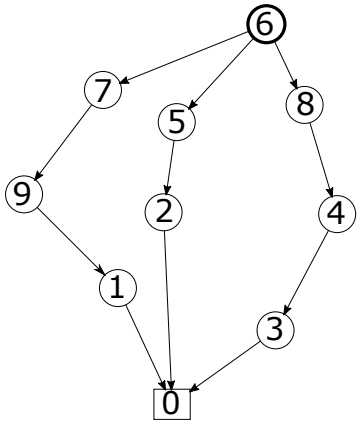
(a) Customer nodes denoted by circles and depot denoted by a square.



(b) After the optimal solution has been found, the length of edge $(i, j) \in E$ is the value of u_{ij}.



(c) The edges (i, j) for which $y_{ij} = 1$ are added with bold lines.



(d) Only the edges (i, j) with $c_{ij} > 0$ are shown. The new starting point is in bold.

Figure 2.2: Route construction with depot location

There are exactly $n^C - n^S Q^S_{min}$ routes of cardinality Q^S_{max}, and $n^S - (n^C - n^S Q^S_{min})$ routes of cardinality Q^S_{min}. If n^C is divisible by n^S, then $Q^S_{min} = Q^S_{max}$.

3. **Lower bound capacity constraint.** Following the idea of subtour elimination and capacity restrictions for the CVRP, an upper bound is usually imposed on a given route to maintain feasibility. However, the ideally balanced routes often require a fractional number of customers. This is then rounded up, and for larger instances, imposing this only as an upper bound can lead to highly unbalanced routes. Enforcing a lower bound is a trickier concept, and for this purpose, it is useful to involve the *third index*, representing each seller. Now we can precisely constrain the lower bound on a specific route, thus maintaining the desired balance.

4. **Adapted MTZ constraints.** The Miller-Tucker-Zemlin (MTZ) constraints are a polynomial number of subtour elimination constraints which were discussed

in Section 1.5. Due to this property they can be explicitly used with LP solvers. Initially introduced for the needs of the TSP, they were eventually generalised to be valid for the asymmetric VRP, taking care of the capacity requirements of the vehicles at the same time as preventing subtours. Since our problem is defined as a symmetric VRP, this is mended by defining a directed graph which assumes that for all pairs of nodes i, j the length of the edge (i, j) is equal to the length of (j, i). Using these constraints we have introduced an *upper bound* to the sizes of the routes. With that, the proper balancing is complete.

2.4.2 Two-index flow model

It is interesting to reduce the number of binary variables in a model as this lowers the dimension of the feasible region. The model from Section 2.4.1 can be simplified by introducing a substantial change to the degree constraints. This is necessary since by locating a new starting point, we are effectively implying that one of the customer nodes will have n^S incident edges, while the rest have two as usual. We propose the following model:

Sets:

N^0: set of customer nodes,

N: $N^0 \cup \{0\}$, set of all nodes.

Parameters:

c^S_{ij}: cost for a seller to travel from node i to node j,

n^C: number of customers,

n^S: number of sellers,

$r^S(V)$: upper bounding function,

$\rho^S(V)$: lower bounding function,

Q^S_{max}: seller route capacity (maximum route length),

Q^S_{min}: minimum seller route length.

Variables:

$$u_{ij} = \begin{cases} 1, & \text{if a seller traverses edge } (i, j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

$$z_i = \begin{cases} 1, & \text{if } i \in N \text{ is the starting point,} \\ 0, & \text{otherwise.} \end{cases}$$

$$\min \sum_{i \in N} \sum_{j \in N} c_{ij}^S u_{ij}$$

$$\text{s.t.} \sum_{i \in N} u_{ij} = 1 - z_j \qquad \forall j \in N, \tag{2.17}$$

$$\sum_{j \in N} u_{ij} = 1 + (n^S - 1) z_i \qquad \forall i \in N, \tag{2.18}$$

$$\sum_{i \in N} u_{i0} = n^S, \tag{2.19}$$

$$\sum_{i \in N} z_i = 1, \tag{2.20}$$

$$\sum_{i \in V} \sum_{j \in V^C \setminus \{0\}} u_{ij} \geq r^S(V) \qquad \forall V \subset N^0, \quad V \neq \emptyset, \tag{2.21}$$

$$\sum_{i \in V} \left(\sum_{j \in V^C \setminus \{0\}} \left(u_{ij} + u_{ji} \right) + z_i \right) \geq \rho^S(V) \qquad \forall V \subset N^0, \quad V \neq \emptyset, \tag{2.22}$$

$$u_{ij} \in \{0, 1\} \qquad \forall i, j \in N, \tag{2.23}$$

$$z_i \in \{0, 1\} \qquad \forall i \in N. \tag{2.24}$$

Here the objective function minimises the total cost of the routes. Constraint (2.17) implies that every customer location is visited by a seller exactly once, unless it is the starting point. Similarly, constraint (2.18) says that every customer location is left by a seller exactly once, except for the case when it is the starting point. In that case, all sellers leave from that customer location and none of them enter it. Constraint (2.19) implies that the depot needs to be visited by all sellers, and constraint (2.20) limits the number of starting points to one. Constraints (2.21) and (2.22) are the *workload balancing* constraints. The right-hand sides are the *upper* and *lower* bounding functions, respectively, defined below. The decision variables are binary, which is imposed by constraints (2.23) and (2.24).

We can now clarify the properties of this model.

1. **Degree constraints where an unknown customer location is a starting point.** These constraints break the rule of creating closed routes for every vehicle ((2.17) and (2.18)). As a result, it is challenging to use some standard solution approaches that rely on the customer nodes' degree being equal to 2. However, these constraints are especially useful for problems involving depot location, where it is to be decided if a given node is a customer or a depot, as they reduce the number of binary variables and constraints needed to formulate the problem.

2. **Maximum and minimum route lengths.** These values are defined as in Section 2.4.1.

3. **Upper bounding function.** The vehicle capacity imposes a known restriction for the CVRP. This is usually defined as

$$\sum_{i \in V} \sum_{j \in V^C} u_{ij} \geq r^S(V), \quad \forall V \subset N^0, \; V \neq \emptyset. \tag{2.25}$$

The left-hand side counts the number of edges that leave from a customer subset V and the right-hand side is the minimum number of vehicles needed to serve all the customers in V. As mentioned in Section 1.5, it is common to use a lower bound of this value defined as $\frac{\text{total demand in } V}{\text{vehicle capacity}}$, or in our case:

$$r^S(V) = \left\lceil \frac{|V|}{Q^S_{max}} \right\rceil, \quad V \subset N^0. \tag{2.26}$$

Even though the degree constraints of the Seller Problem differ from the CVRP, constraint (2.25) remains valid. We can illustrate this with the small example in Figure 2.3. In (a) a feasible solution S_1 is shown for $Q^S_{max} = 3$. Constraint (2.25) is satisfied for the set V_1 shown with a dashed line in (b). The constraint is also satisfied for every subset of V_1. A second solution S_2 for $Q^S_{max} = 3$ is shown in (c), which is infeasible as the only route shown is of length greater than 3. Constraint (2.25) is satisfied for V_2 shown in (d). However, the set $V_2 \backslash \{sp\}$ where sp is the starting point shown in red, *violates* the upper bounding constraint. No feasible solution is violated by constraints (2.25).

4. **Lower bounding function.** Workload balancing by cardinality is a concept that is not as common in the literature as the capacitated version of the VRP, where the route sizes are bounded by the vehicle capacity, time duration or cost. In those cases, it is often only the upper bound of the route size that is of interest. Here we are looking to impose a lower bound on the number of customers visited by each seller. In the case of the previous model (Section 2.4.1) this was easier as the routes were indexed by seller, and this restriction was given as (2.11). However, it not possible to impose a lower bound on the number of edges that enter into a node subset, or that connect nodes within the subset. For this reason we are only controlling whether for a given customer subset V there needs to be an edge crossing its boundary, disregarding the depot.

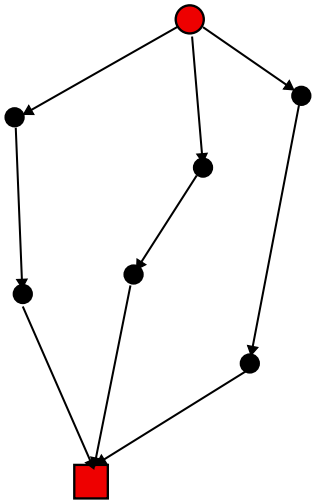
   The left-hand side of the lower bounding constraint counts the number of edges that cross the boundary of a nonempty set $V \subseteq N^0$, ignoring the edges directly connected to the depot: $\sum_{i \in V} \sum_{j \in V^C \backslash \{0\}} (u_{ij} + u_{ji})$. The right-hand side decides whether or not a subset of size $|V|$ requires *at least one* edge to cross its boundary, or no restriction is needed. If no restriction is needed, the left-hand side is simply non-negative as a sum of binary variables (2.23).

   A restriction is imposed only if the cardinality of the set V is smaller than Q^S_{min}, the minimal length of any route. We consider the expression $\left\lfloor \frac{Q^S_{min}}{|V|} \right\rfloor$, for which only the following two cases are possible:

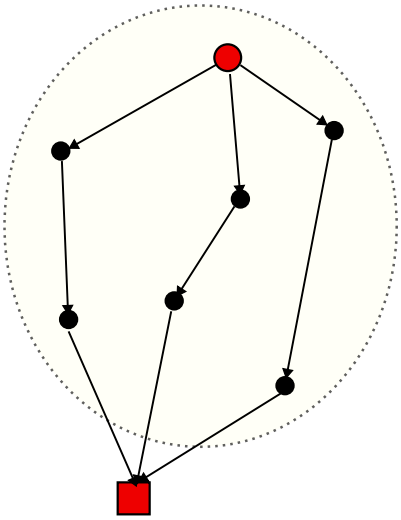
   a) $\left\lfloor \frac{Q^S_{min}}{|V|} \right\rfloor = 0 \iff |V| > Q^S_{min}$,

   b) $\left\lfloor \frac{Q^S_{min}}{|V|} \right\rfloor > 0 \iff |V| \leq Q^S_{min}$.

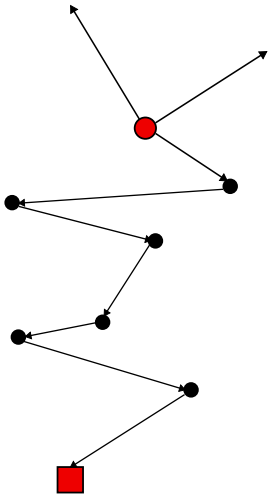
   As we cannot impose a lower bound greater than 1 to $\sum_{i \in V} \sum_{j \in V^C \backslash \{0\}} (u_{ij} + u_{ji})$ for an arbitrary set V, we will restrict the case b) to a maximum value of 1.



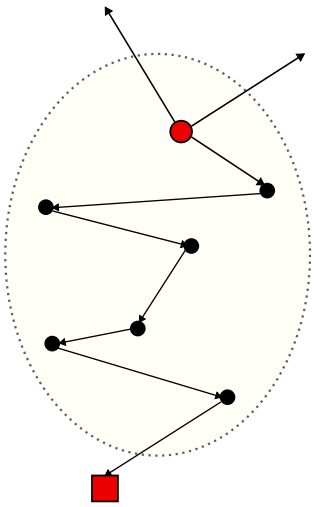
(a) Solution S_1.



(b) Subset $V_1 \subseteq N$ shown with dashed lines.



(c) Solution S_2.



(d) Subset $V_2 \subseteq N$ shown with dashed lines.

Figure 2.3: Toy examples for workload balance for the seller problem.

Given the above, the lower bound workload balancing constraint takes the form

$$\sum_{i \in V} \sum_{j \in V^C \setminus \{0\}} (u_{ij} + u_{ji}) \geq \min \left\{ \left\lfloor \frac{Q^S_{min}}{|V|} \right\rfloor, 1 \right\}. \tag{2.27}$$

Let us now consider the location of the starting point. From the example in Figure 2.3 (b), consider the set V_1 shown with a dashed line and let $Q^S_{min} = 6$. The inequality (2.27) is $0 \geq \min\left\{\left\lfloor \frac{6}{7} \right\rfloor, 1\right\} = 0$, which means that V_1 *satisfies* the constraint even though S_1 is infeasible. Now consider a small example with $n^S = 1$ and a single feasible route. The route starting at sp will violate (2.27) with $0 \geq 1$ as no edges exit the starting point.

For this reason we will alter (2.27) by adding a *big M* to the left-hand side if V

contains a starting point.

$$\sum_{i\in V}\left(\sum_{j\in V^C\setminus\{0\}}(u_{ij}+u_{ji})+Mz_i\right)\geq \min\left\{\left\lfloor\frac{Q^S_{min}}{|V|}\right\rfloor,1\right\}. \tag{2.28}$$

Since the right-hand side can be at most 1, it is sufficient to take $M=1$.

$$\rho^S(V)=\min\left\{\left\lfloor\frac{Q^S_{min}}{|V|}\right\rfloor,1\right\},\quad V\subseteq N^0.$$

$$\sum_{i\in V}\left(\sum_{j\in V^C\setminus\{0\}}(u_{ij}+u_{ji})+z_i\right)\geq \rho^S(V),\quad V\subseteq N^0. \tag{2.29}$$

Proposition 2.4.1. *Constraint* (2.27) *enforces the lower bound of route cardinality for a VRP with workload balance by cardinality.*

Proof. The proof follows directly from the explanation above. □

Proposition 2.4.2. *Constraint* (2.29) *enforces the lower bound of route cardinality for the Seller Problem.*

Proof. The proof follows directly from the explanation above. □

Both models introduced above can be easily adjusted to a multi-period case by adding an additional index to the decision variables. This will be shown in the complete model.

2.5 The truck problem

The second stage of the problem consists of the trucks' operation. Similarly to the case of the seller problem, we will simplify this sub-problem by removing the multi-period property, and we will consider only the operation during a single period (day). This sub-problem is a *VRP with workload balance*, which we call the truck problem.

We define a routing problem for a set of *trucks*, *customers*, and a fixed *depot*. All routes start and finish at the depot. The customers must be visited exactly once and the cardinality of the truck routes are restricted by *workload balance*.

The models proposed in Section 2.4 will now be adjusted to the truck problem.

2.5.1 Three-index flow model

We define the following notation:

Sets:

N^0: set of customer nodes,

N: $N^0\cup\{0\}$, set of all nodes.

Parameters:

c_{ij}^T : cost for a truck to travel from node i to node j,

n^C : number of customers,

n^T : number of trucks,

Q_{max}^T : truck route capacity (maximum route length),

Q_{min}^T : minimum truck route length.

Variables:

$$x_{ij}^k = \begin{cases} 1, & \text{if truck } k \text{ traverses edge } (i,j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

θ_i : auxiliary continuous variable for $i \in N$.

The travelling cost is defined by the cost function $c^T : E \to \mathbb{R}_0^+$. The x-variables define the closed routes for the trucks, starting and ending at the depot. The complete formulation is given below:

$$\min \sum_{i,j \in N} c_{ij}^T \sum_{k=1}^{n^T} x_{ij}^k$$

$$\text{s.t.} \sum_{i \in N} \sum_{k=1}^{n^T} x_{ij}^k = 1 \qquad \forall j \in N^0, \tag{2.30}$$

$$\sum_{i \in N} \sum_{k=1}^{n^T} x_{ji}^k = 1 \qquad \forall j \in N^0, \tag{2.31}$$

$$\sum_{i \in N} \sum_{k=1}^{n^T} x_{i0}^k = n^T, \tag{2.32}$$

$$\sum_{j \in N^0} \sum_{k=1}^{n^T} x_{0j}^k = n^T, \tag{2.33}$$

$$\sum_{i \in N} \sum_{j \in N} x_{ij}^k \geq Q_{min}^T \qquad \forall k = 1, \ldots, n^T, \tag{2.34}$$

$$\theta_j - \theta_i + Q_{max}^T \sum_{k=1}^{n^T} x_{ij}^k \leq Q_{max}^T - 1 \qquad \forall i, j \in N^0, i \neq j, \tag{2.35}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall i, j \in N, \quad k \in \{1 \ldots n^T\}, \tag{2.36}$$

$$1 \leq \theta_i \leq Q_{max}^T \qquad \forall i \in N. \tag{2.37}$$

The objective function minimises the total travelling cost for all trucks. Constraints (2.30) – (2.33) are the degree constraints for all nodes, including the depot. The workload balance is imposed by constraints (2.34) and (2.35). The former controls the route cardinality lower bound. The latter is the MTZ constraint for subtour elimination and route cardinality bound (Section 2.4.1). The bounds on the decision

variables are given by (2.36) and (2.37).

Similarly as in Section 2.4.1, the maximum and minimum route lengths for each truck are defined as:

$$Q^T_{max} = \left\lceil \frac{n^C}{n^T} \right\rceil, \qquad Q^T_{min} = \left\lfloor \frac{n^C}{n^T} \right\rfloor. \tag{2.38}$$

2.5.2 Two-index flow model

As in the case of the seller problem, we can reduce the number of binary decision variables if we formulate the problem using the standard two-index model for the CVRP. Unlike the seller problem, here the degree constraints are not an obstacle and this model will not deviate from its classic formulation in that sense. However, the workload balance by cardinality forces an additional set of constraints to control the lower bound on the route size.

Sets:

N^0: set of customers nodes,

N: $N^0 \cup \{0\}$, set of all nodes.

Parameters:

c^T_{ij}: cost of travelling from node i to node j,

n^C: number of customers,

n^T: number of trucks,

$r^T(V)$: upper bounding function for trucks,

$\rho^T(V)$: lower bounding function for trucks,

Q^T_{max}: truck route capacity (maximum route length),

Q^T_{min}: minimum truck route length.

Variables:

$$x_{ij} = \begin{cases} 1, & \text{if a truck traverses edge } (i,j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

$$\min \sum_{i \in N} \sum_{j \in N} c^T_{ij} x_{ij}$$

$$\text{s.t.} \sum_{i \in N} x_{ij} = 1 \qquad \forall j \in N^0, \tag{2.39}$$

$$\sum_{j \in N} x_{ij} = 1 \qquad \forall i \in N^0, \tag{2.40}$$

$$\sum_{i \in N^0} x_{i0} = n^T, \tag{2.41}$$

$$\sum_{j \in N^0} x_{0j} = n^T, \tag{2.42}$$

$$\sum_{i \in V} \sum_{j \in V^C \setminus \{0\}} x_{ij} \geq r^T(V) \qquad \forall V \subset N^0, \quad V \neq \emptyset, \tag{2.43}$$

$$\sum_{i \in V} \sum_{j \in V^C \setminus \{0\}} (x_{ij} + x_{ji}) \geq \rho^T(V) \qquad \forall V \subset N^0, \quad V \neq \emptyset, \tag{2.44}$$

$$x_{ij} \in \{0,1\} \qquad \forall i, j \in N. \tag{2.45}$$

The objective function minimises the total cost of the truck routes. Constraints (2.39) – (2.42) are the standard degree constraints for all customer nodes and the depot, as in the CVRP. Constraints (2.43) and (2.44) impose upper and lower bounds to the route cardinality, respectively.

The right-hand side of constraint (2.43) is the upper bounding function, introduced in Section 2.4.2. This function is equivalent to (2.26), and, using the corresponding notation for the truck problem and (2.38), it takes the form:

$$r^T(V) = \left\lceil \frac{|V|}{Q^T_{max}} \right\rceil, \quad V \subset N^0.$$

The right-hand side of constraint (2.44) is the lower bounding function, (2.27). With the corresponding notation and (2.38) it becomes:

$$\sum_{i \in V} \sum_{j \in V^C \setminus \{0\}} (x_{ij} + x_{ji}) \geq \min \left\{ \left\lfloor \frac{Q^T_{min}}{|V|} \right\rfloor, 1 \right\}.$$

2.6 Two-stage multi-period vehicle routing problem with depot location and workload balance

In the previous sections the problem at hand was separated and studied as two sub-problems, the seller problem and the truck problem. The main goal of this chapter is to solve the two sub-problems simultaneously over a rolling time horizon.

Converting the two and three-index models above to fit a multi-period scenario is simple. It is done by introducing an additional index to the decision variables representing the period (in our case a period is a day). Each customer is visited once in the planning horizon (in our case the planning horizon is a working week) and all constraints are adjusted accordingly. This change will be shown in the formulations proposed in the remainder of the section. We will combine the three and two-index models from Section 2.4 and Section 2.5, and with the multi-period property, they become *four* and *three-index* models, respectively.

2.6.1 Four-index flow model

Combining the models from Section 2.4.1 and Section 2.5.1, we define the following notation:

Sets:

N^0: set of customer nodes,
N: $N^0 \cup \{0\}$, set of all nodes,
D: planning horizon.

Parameters:

c_{ij}^S: cost for a seller to travel from node i to node j,
c_{ij}^T: cost for a truck to travel from node i to node j,
n^D: number of days,
n^C: number of customers,
n^S: number of sellers,
n^T: number of trucks,
Q_{max}^S: seller route capacity (maximum route length),
Q_{min}^S: minimum seller route length,
Q_{max}^T: truck route capacity (maximum route length),
Q_{min}^T: minimum truck route length.

Variables:

$$x_{ij}^{kd} = \begin{cases} 1, & \text{if truck } k \text{ traverses edge } (i,j) \in E \text{ on day } d, \\ 0, & \text{otherwise.} \end{cases}$$

$$u_{ij}^{kd} = \begin{cases} 1, & \text{if seller } k \text{ traverses edge } (i,j) \in E \text{ on day } d, \\ 0, & \text{otherwise.} \end{cases}$$

$$z_i^d = \begin{cases} 1, & \text{if } i \in N \text{ is the starting point for the sellers on day } d, \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{ij}^d = \begin{cases} 1, & \text{if } i \text{ is the common starting point on day } d \\ & \text{and } j \text{ is the first customer visited by some seller,} \\ 0, & \text{otherwise.} \end{cases}$$

σ_i: auxiliary continuous variable for $i \in N$,
θi: auxiliary continuous variable for $i \in N$.

The n^T trucks and the n^S sellers are routed independently, but a connection is established by the days (every customer is visited by a seller on day d and by a truck on day $d+1$). The set of n^C customers has to be split such that approximately the same number of customers is visited every day of the planning horizon D, and every customer is visited exactly once by a seller and once by a truck. Both the seller and the truck routings are performed on the same subsets of customers, and there are

n^D of these disjoint subsets. The cost function for the seller routes is $c^S : E \to \mathbb{R}_0^+$ and the cost for the truck routes is $c^T : E \to \mathbb{R}_0^+$. We maintain this difference in notation in line with the possibility that the trucks and the sellers may have different travel costs, e.g. different fuel consumption or different travel time.

As in the three-index formulation for the seller problem (Section 2.4.1), we define:

$$c_{0i}^S = 0 \quad \forall i \in N.$$

The problem is then formulated as follows:

$$\min \sum_{i,j \in N} \sum_{d \in D} \left(c_{ij}^T \sum_{k=1}^{n^T} x_{ij}^{kd} + c_{ij}^S \left(\sum_{k=1}^{n^S} u_{ij}^{kd} + y_{ij}^d \right) \right)$$

$$\text{s.t.} \sum_{j \in N} \sum_{k=1}^{n^T} \sum_{d \in D} x_{ij}^{kd} = 1 \qquad \forall i \in N^0, \tag{2.46}$$

$$\sum_{i \in N} \sum_{k=1}^{n^T} \sum_{d \in D} x_{ij}^{kd} = 1 \qquad \forall j \in N^0, \tag{2.47}$$

$$\sum_{j \in N} \sum_{k=1}^{n^S} \sum_{d \in D} u_{ij}^{kd} = 1 \qquad \forall i \in N^0, \tag{2.48}$$

$$\sum_{i \in N} \sum_{k=1}^{n^S} \sum_{d \in D} u_{ij}^{kd} = 1 \qquad \forall j \in N^0, \tag{2.49}$$

$$\sum_{i \in N} \left(x_{ij}^{kd} - x_{ji}^{kd} \right) = 0 \qquad \forall j \in N,\ \forall d \in D,\ \forall k = 1, \ldots, n^T, \tag{2.50}$$

$$\sum_{i \in N} \left(u_{ij}^{kd} - u_{ji}^{kd} \right) = 0 \qquad \forall j \in N,\ \forall d \in D,\ \forall k = 1, \ldots, n^S, \tag{2.51}$$

$$\sum_{i \in N} \sum_{k=1}^{n^T} x_{i0}^{kd} = n^T \qquad \forall d \in D, \tag{2.52}$$

$$\sum_{i \in N} \sum_{k=1}^{n^S} u_{i0}^{kd} = n^S \qquad \forall d \in D, \tag{2.53}$$

$$\sum_{j \in N} \sum_{k=1}^{n^T} x_{0j}^{kd} = n^T \qquad \forall d \in D, \tag{2.54}$$

$$\sum_{j \in N} \sum_{k=1}^{n^S} u_{0j}^{kd} = n^S \qquad \forall d \in D, \tag{2.55}$$

$$\sum_{i \in N} z_i^d = 1 \qquad \forall d \in D, \tag{2.56}$$

$$\sum_{j \in N} y_{ij}^d = n^S z_i^d \qquad \forall d \in D,\ \forall i \in N, \tag{2.57}$$

$$\sum_{i \in N} y_{ij}^{d} = \sum_{k=1}^{n^S} u_{0j}^{kd} \qquad \forall d \in D,\ \forall j \in N, \tag{2.58}$$

$$\sum_{j \in N} \sum_{k=1}^{n^S} u_{ij}^{kd} - \sum_{j \in N} \sum_{k=1}^{n^T} x_{ij}^{k,d+1} = 0 \qquad \forall i \in N^0,\ \forall d \in \{1, \ldots, n^D - 1\}, \tag{2.59}$$

$$\sum_{j \in N} \sum_{k=1}^{n^S} u_{ij}^{k,n^D} - \sum_{j \in N} \sum_{k=1}^{n^T} x_{ij}^{k,1} = 0 \qquad \forall i \in N^0, \tag{2.60}$$

$$\sum_{i \in N} \sum_{j \in N} x_{ij}^{kd} \geq Q_{min}^{T} \qquad \forall d \in D,\ k = 1, \ldots, n^T, \tag{2.61}$$

$$\sum_{i \in N} \sum_{j \in N} u_{ij}^{kd} \geq Q_{min}^{S} \qquad \forall d \in D,\ k = 1, \ldots, n^S, \tag{2.62}$$

$$\theta_j - \theta_i + Q_{max}^{T} \sum_{d \in D} \sum_{k=1}^{n^T} x_{ij}^{kd} \leq Q_{max}^{T} - 1 \qquad \forall i, j \in N^0,\ i \neq j, \tag{2.63}$$

$$\sigma_j - \sigma_i + Q_{max}^{S} \sum_{d \in D} \sum_{k=1}^{n^S} u_{ij}^{kd} \leq Q_{max}^{S} - 1 \qquad \forall i, j \in N^0,\ i \neq j, \tag{2.64}$$

$$1 \leq \theta_i \leq Q_{max}^{T}, \quad 1 \leq \sigma_i \leq Q_{max}^{S} \qquad \forall i \in N, \tag{2.65}$$

$$x_{ij}^{kd}, u_{ij}^{ld}, y_{ij}^{d}, z_{i}^{d} \in \{0, 1\} \qquad \forall i, j \in N,\ d \in D,\ k = 1, \ldots, n^T,\ l = 1, \ldots, n^S. \tag{2.66}$$

In the formulation above, constraints (2.46)-(2.49) are the customer node degree constraints for both the seller and the truck routes. Constraints (2.50) and (2.51) impose that a seller and a truck, respectively, arrive at a customer location and leave on the same day. Constraints (2.52)-(2.55) are the depot constraints – meaning that exactly n^T trucks and exactly n^S sellers leave from and arrive at the depot. Constraint (2.56) means that for every day there is exactly one starting point for the sellers. Constraints (2.57) and (2.58) define the first customer that each seller visits on their route. All of these constraints are merely extensions to the ones introduced in Section 2.4.1 and Section 2.5.1.

The main connection between the seller and the truck problem is established by Constraints (2.59) and (2.60). They enforce that every customer that is visited by a seller on a given day is in turn visited by a truck on the following day. Since this is a rolling time horizon, if day d is the last day of the horizon, n^D, then day $d+1$ is day 1.

As previously explained, the workload balance by cardinality is enforced by (2.61) – (2.64). The binary decision variables are defined with (2.66), and the continuous variables necessary for the MTZ constraints are given with (2.65).

The upper and lower bounding functions from the workload balance constraints are fully defined in Section 2.4.2 and Section 2.5.2 for the single period simplification. For this case their definition remains the same, while the existence of multiple periods is factored into the calculation of the maximum and minimum route lengths:

$$Q_{max}^{S} = \left\lceil \frac{n^C}{n^D n^S} \right\rceil,\ Q_{min}^{S} = \left\lfloor \frac{n^C}{n^D n^S} \right\rfloor,\ Q_{max}^{T} = \left\lceil \frac{n^C}{n^D n^T} \right\rceil,\ Q_{min}^{T} = \left\lfloor \frac{n^C}{n^D n^T} \right\rfloor. \tag{2.67}$$

2.6.2 Simplification and redundant constraints

The nature of this problem unites two very similar and seemingly independent problems – the truck and seller problems – into one large problem, connected only through the additional constraint that is the planning time horizon. This connection allows for some of the original constraints to be removed while maintaining feasibility of the problem. In this section we will analyse the model from Section 2.6.1 to show that several of the original constraints are redundant.

Constraint (2.46) can be obtained from (2.48), (2.59) and (2.60):

$$\sum_{j\in N}\sum_{k=1}^{n^S} u_{ij}^{kd} - \sum_{j\in N}\sum_{k=1}^{n^T} x_{ij}^{k,d+1} = 0 \quad \forall i \in N, \forall d \in D\backslash\{n^D\}$$

$$\Longrightarrow \sum_{d\in D}\left(\sum_{j\in N}\sum_{k=1}^{n^S} u_{ij}^{kd} - \sum_{j\in N}\sum_{k=1}^{n^T} x_{ij}^{k,d}\right) = 0 \quad \forall i \in N$$

$$\overset{(2.46)}{\Longrightarrow} \sum_{j\in N}\sum_{k=1}^{n^T}\sum_{d\in D} x_{ij}^{kd} = \sum_{j\in N}\sum_{k=1}^{n^S}\sum_{d\in D} u_{ij}^{kd} = 1.$$

From here it follows that constraint (2.47) is implied by the above expression and by constraint (2.50). Similarly, constraint (2.49) comes from (2.48) and (2.51):

$$\sum_{i\in N}\left(u_{ij}^{kd} - u_{ji}^{kd}\right) = 0 \quad \forall j \in N,\ \forall d \in D,\ \forall k = 1,\dots,n^S$$

$$\Longrightarrow \sum_{d\in D}\left(\sum_{i\in N}\left(u_{ij}^{kd} - u_{ji}^{kd}\right)\right) = 0 \quad \forall j \in N,\ \forall k = 1,\dots,n^S$$

$$\Longrightarrow \sum_{k=1}^{n^S}\left(\sum_{i\in N}\sum_{d\in D}\left(u_{ij}^{kd} - u_{ji}^{kd}\right)\right) = 0 \quad \forall j \in N$$

$$\overset{(2.48)}{\Longrightarrow} \sum_{i\in N}\sum_{k=1}^{n^S}\sum_{d\in D} u_{ji}^{d} = \sum_{i\in N}\sum_{k=1}^{n^S}\sum_{d\in D} u_{ij}^{d} = 1 \quad \forall j \in N.$$

Constraint (2.52) is implied by (2.50) and (2.54):

$$\sum_{i\in N}\left(x_{ij}^{kd} - x_{ji}^{kd}\right) = 0 \quad \forall j \in N,\ \forall d \in D,\ \forall k = 1,\dots,n^T,$$

$$\Longrightarrow \sum_{k=1}^{n^T}\left(\sum_{i\in N}\left(x_{ij}^{kd} - x_{ji}^{kd}\right)\right) = 0 \quad \forall j \in N,\ \forall d \in D$$

$$\Longrightarrow \sum_{i\in N}\sum_{k=1}^{n^T} x_{ij}^{kd} = \sum_{i\in N}\sum_{k=1}^{n^T} x_{ji}^{kd} \quad \forall j \in N,\ \forall d \in D$$

$$\overset{(2.54)}{\Longrightarrow} \sum_{i\in N}\sum_{k=1}^{n^T} x_{i0}^{kd} = \sum_{i\in N}\sum_{k=1}^{n^T} x_{0i}^{kd} = n^T \quad \forall d \in D.$$

Constraint (2.55) is implied by (2.56), (2.57) and (2.58):

$$\sum_{k=1}^{n^S} u_{0j}^d = \sum_{i \in N} y_{ij}^d \quad \forall j \in N, \forall d \in D$$

$$\implies \sum_{j \in N} \left(\sum_{k=1}^{n^S} u_{0j}^d - \sum_{i \in N} y_{ij}^d \right) = 0 \quad \forall d \in D$$

$$\overset{(2.57)}{\implies} \sum_{j \in N} \sum_{k=1}^{n^S} u_{0j}^d = \sum_{j \in N} \sum_{i \in N} y_{ij}^d = \sum_{i \in N} \left(\sum_{j \in N} y_{ij}^d \right)$$

$$= \sum_{i \in N} \sum_{k=1}^{n^S} n^S z_i^d \quad \forall d \in D$$

$$\overset{(2.56)}{\implies} \sum_{j \in N} \sum_{k=1}^{n^S} u_{0j}^d = n^S \sum_{i \in N} z_i^d = n^S \quad \forall d \in D.$$

Constraint (2.53) comes from the above expression and (2.51).
Therefore, we can now rewrite the model as follows:

$$\min \sum_{i,j \in N} \sum_{d \in D} \left(c_{ij}^T \sum_{k=1}^{n^T} x_{ij}^{kd} + c_{ij}^S \left(\sum_{k=1}^{n^S} u_{ij}^{kd} + y_{ij}^d \right) \right)$$

$$\text{s.t.} \sum_{j \in N} \sum_{d \in D} \sum_{k=1}^{n^S} u_{ij}^{kd} = 1 \qquad \forall i \in N,$$

$$\sum_{i \in N} \left(u_{ij}^{kd} - u_{ji}^{kd} \right) = 0 \qquad \forall j \in N, \forall d \in D, k = 1, \ldots, n^S,$$

$$\sum_{i \in N} \left(x_{ij}^{kd} - x_{ji}^{kd} \right) = 0 \qquad \forall j \in N, \forall d \in D, k = 1, \ldots, n^T,$$

$$\sum_{j \in N} \sum_{k=1}^{n^T} x_{0j}^{kd} = n^T \qquad \forall d \in D,$$

$$\sum_{i \in N} z_i^d = 1 \qquad \forall d \in D,$$

$$\sum_{j \in N} y_{ij}^d = n^S z_i^d \qquad \forall d \in D, \forall i \in N,$$

$$\sum_{i \in N} y_{ij}^d = \sum_{k=1}^{n^S} u_{0j}^{kd} \qquad \forall d \in D, \forall j \in N,$$

$$\sum_{j \in N} \sum_{k=1}^{n^S} u_{ij}^{kd} - \sum_{j \in N} \sum_{k=1}^{n^T} x_{ij}^{k(d+1)} = 0 \qquad \forall i \in N^0, \forall d \in \{1, \ldots, n^D - 1\},$$

$$\sum_{j \in N} \sum_{k=1}^{n^S} u_{ij}^{kn^D} - \sum_{j \in N} \sum_{k=1}^{n^T} x_{ij}^{k0} = 0 \qquad \forall i \in N^0,$$

$$\sum_{i \in N} \sum_{j \in N} x_{ij}^{kd} \geq Q_{min}^T \qquad \forall d \in D, k = 1, \ldots, n^T,$$

$$\sum_{i\in N}\sum_{j\in N} u_{ij}^{kd} \geq Q_{min}^{S} \qquad \forall d \in D, k = 1,\ldots,n^{S},$$

$$\theta_j - \theta_i + Q_{max}^{T} \sum_{d\in D}\sum_{k=1}^{n^T} x_{ij}^{kd} \leq Q_{max}^{T} - 1 \qquad \forall i,j \in N^0, i \neq j,$$

$$\sigma_j - \sigma_i + Q_{max}^{S} \sum_{d\in D}\sum_{k=1}^{n^S} u_{ij}^{kd} \leq Q_{max}^{S} - 1 \qquad \forall i,j \in N^0, i \neq j,$$

$$x_{ij}^{kd}, u_{ij}^{ld}, y_{ij}^{d}, z_{i}^{d} \in \{0,1\} \qquad \forall i,j \in N, d \in D, k = 1,\ldots,n^{T}, l = 1,\ldots,n^{S},$$

$$1 \leq \theta_i \leq Q_{max}^{T}, \quad 1 \leq \sigma_i \leq Q_{max}^{S} \qquad \forall i \in N.$$

2.6.3 Three-index flow model

In this section we will unify the two-index formulations for the seller and truck problem detailed in Section 2.4.2 and Section 2.5.2, using the simplification proposed in Section 2.6.2 to formulate the complete problem.

Sets:

N^0: set of customer nodes,

N: $N^0 \cup \{0\}$, set of all nodes,

D: planning horizon.

Parameters:

c_{ij}^{S}: cost for a seller to travel from node i to node j,

c_{ij}^{T}: cost for a truck to travel from node i to node j,

n^{D}: number of days,

n^{C}: number of customers,

n^{S}: number of sellers,

n^{T}: number of trucks,

Q_{max}^{S}: seller route capacity (maximum route length),

Q_{min}^{S}: minimum seller route length,

Q_{max}^{T}: truck route capacity (maximum route length),

Q_{min}^{T}: minimum truck route length,

$r^{S}(V)$: upper bounding function for sellers,

$\rho^{S}(V)$: lower bounding function for sellers,

$r^{T}(V)$: upper bounding function for trucks,

$\rho^{T}(V)$: lower bounding function for trucks.

Variables:

$$x_{ij}^d = \begin{cases} 1, & \text{if a truck traverses edge } (i,j) \in E \text{ on day } d, \\ 0, & \text{otherwise.} \end{cases}$$

$$u_{ij}^d = \begin{cases} 1, & \text{if a seller traverses edge } (i,j) \in E \text{ on day } d, \\ 0, & \text{otherwise.} \end{cases}$$

$$z_i^d = \begin{cases} 1, & \text{if } i \in N \text{ is the starting point for the sellers on day } d, \\ 0, & \text{otherwise.} \end{cases}$$

$$\min \sum_{i,j \in N} c_{ij}^T \sum_{d \in D} x_{ij}^d + \sum_{i,j \in N} c_{ij}^S \sum_{d \in D} u_{ij}^d \tag{2.68}$$

$$\text{s.t.} \sum_{d \in D} \left(\sum_{j \in N} u_{ij}^d - \left(n^S - 1\right) z_i^d \right) = 1 \qquad \forall i \in N^0, \tag{2.69}$$

$$\sum_{d \in D} \left(\sum_{i \in N^0} u_{ij}^d + z_j^d \right) = 1 \qquad \forall j \in N^0, \tag{2.70}$$

$$\sum_{j \in N} \left(u_{ij}^d - u_{ji}^d \right) = n^S z_i^d \qquad \forall i \in N^0,\ \forall d \in D, \tag{2.71}$$

$$\sum_{j \in N} \left(x_{ij}^d - x_{ji}^d \right) = 0 \qquad \forall i \in N,\ \forall d \in D, \tag{2.72}$$

$$\sum_{j \in N^0} x_{0j}^d = n^T \qquad \forall d \in D, \tag{2.73}$$

$$\sum_{i \in N} z_i^d = 1 \qquad \forall d \in D, \tag{2.74}$$

$$\sum_{j \in N} \left(u_{ij}^d - x_{ij}^{d+1} \right) = \left(n^S - 1\right) z_i^d \qquad \forall i \in N^0,\ \forall d \in \{1, \ldots, |D| - 1\}, \tag{2.75}$$

$$\sum_{j \in N} \left(u_{ij}^{n^D} - x_{ij}^0 \right) = \left(n^S - 1\right) z_i^{n^D} \qquad \forall i \in N^0, \tag{2.76}$$

$$\sum_{d \in D} \sum_{i \in V} \sum_{j \in V^C \setminus \{0\}} x_{ij}^d \geq r^T(V) \qquad \forall V \subseteq N^0,\ V \neq \emptyset, \tag{2.77}$$

$$\sum_{d \in D} \sum_{i \in V} \sum_{j \in V^C \setminus \{0\}} u_{ij}^d \geq r^S(V) \qquad \forall V \subseteq N^0,\ V \neq \emptyset, \tag{2.78}$$

$$\sum_{d \in D} \sum_{i \in V} \sum_{j \in V^C \setminus \{0\}} (x_{ij}^d + x_{ji}^d) \geq \rho^T(V) \qquad \forall V \subseteq N^0,\ V \neq \emptyset, \tag{2.79}$$

$$\sum_{d \in D} \sum_{i \in V} \left(\sum_{j \in V^C \setminus \{0\}} (u_{ij}^d + u_{ji}^d) + z_i^d \right) \geq \rho^S(V) \qquad \forall V \subseteq N^0,\ V \neq \emptyset, \tag{2.80}$$

$$x_{ij}^d, u_{ij}^d, z_i^d \in \{0,1\} \qquad \forall i, j \in N,\ d \in D. \tag{2.81}$$

Constraints (2.69) – (2.73) are the degree constraints for all customer locations and the fixed depot. The starting point is determined by using (2.74). The connection between the seller and truck problem is established by (2.75) and (2.76). The workload balance constraints are (2.77) – (2.80), and the decision variables are defined in (2.81).

The upper and lower bounding functions from the workload balance constraints are as described in Sections 2.4.2 and 2.5.2.

2.7 Valid inequalities

In this section we introduce a generalisation of the *two-matching* and *comb inequalities* reviewed in Section 1.6. The comb inequalities, a generalisation of the two-matching inequalities, were originally defined as valid inequalities for the TSP and widely used in efficient branch-and-cut algorithms for the majority of the best-known solutions of benchmark TSP instances. They were later generalised to fit the CVRP and recently strengthened and improved. Here we propose a further generalisation of the above mentioned inequalities, valid for the *VRP with workload balance and depot location* (the seller problem, Section (2.4)).

The known VRP generalisations of the two-matching and comb inequalities are valid for the truck problem (Section 2.5). In order to make this distinction and maintain uniformity, we will use the same notation from the seller problem, deviating from the classic x notation for the routing decision variables. What will be discussed next is the *multi-period* case for the seller problem, which can be easily reduced to a single period case by setting the planning horizon D equal to $\{1\}$. We will use the usual notation for the sum of the variables u at the co-boundary of a given subset $V \subseteq N$: $u(\delta(V)) = \sum_{d\in D}\sum_{i\in V, j\notin V}(u^d_{ij} + u^d_{ji})$. The original inequality for the TSP using this notation is

$$u(\delta(H)) + \sum_{j=1}^{t} u(\delta(T_j) \geq 3t + 1 \tag{2.82}$$

and whether it is a 2-matching or comb inequality is determined by the following conditions:

1. **Two-matching inequalities.**

$$|H \cap T_i| = 1, \qquad i = 1,\ldots,t, \tag{2.83}$$
$$|T_i \setminus H| = 1, \qquad i = 1,\ldots,t, \tag{2.84}$$
$$|T_i \cap T_j| = 0, \qquad 1 \leq i \leq j \leq t, \tag{2.85}$$
$$t = 2n+1, \qquad n \in \mathbb{Z}^+. \tag{2.86}$$

2. **Comb inequalities.**

$$|H \cap T_i| \geq 1, \qquad i = 1,\ldots,t, \tag{2.87}$$
$$|T_i \setminus H| \geq 1, \qquad i = 1,\ldots,t, \tag{2.88}$$
$$|T_i \cap T_j| = 0, \qquad 1 \leq i \leq j \leq t, \tag{2.89}$$
$$t = 2n+1, \qquad n \in \mathbb{Z}^+. \tag{2.90}$$

2.7.1 Generalised two-matching inequalities

We define sets $H, T_1, ..., T_t \subseteq N^0$ that satisfy conditions (2.83) – (2.86). For a given $k \in \{1, \dots, t\}$, let us define nodes $i_{1k} = H \cap T_k$ and $i_{2k} = T_k \setminus H$. Following (2.83) and (2.84), these nodes exist and are unique. The *generalised 2-matching inequality* takes an additional non-negative term on the right-hand side, depending on the location of the starting point in each time period.

$$u(\delta(H)) + \sum_{j=1}^{t} u(\delta(T_j)) \geq 3t + \frac{1}{2}(n^S - 2) \sum_{d \in D} \sum_{k=1}^{t} \left(z^d_{i_{1k}} + z^d_{i_{2k}} \right).$$

When neither of the nodes is a starting point, the inequality becomes the classic 2-matching constraint as the additional term is zero. As the left-hand side of the inequality is always an even integer and $3t$ is always an odd number because t is odd by definition, the right-hand side can be increased by 1 as in (2.82). Therefore, our inequality can be further strengthened by adding either the new term or 1, whichever is greater:

$$u(\delta(H)) + \sum_{j=1}^{t} u(\delta(T_j)) \geq 3t + max \left\{ \frac{1}{2}(n^S - 2) \sum_{d \in D} \sum_{k=1}^{t} \left(z^d_{i_{1k}} + z^d_{i_{2k}} \right), 1 \right\}.$$

Proposition 2.7.1. *The generalised 2-matching inequalities are valid for the multi-period seller problem.*

Proof. We claim that

$$u(\delta(H)) + \sum_{j=1}^{t} u(\delta(T_j)) \geq \frac{1}{2} \left[\sum_{j=1}^{t} u(\delta(H \cap T_j)) + \sum_{j=1}^{t} u(\delta(T_j \setminus H)) + \sum_{j=1}^{t} u(\delta(T_j)) \right]. \quad (2.91)$$

The inequality above follows from the fact that each co-boundary in the square brackets on the right-hand side is counted twice, so it is halved. In the case of 2-matching inequalities, $|T_j| = 2$ for every $j = 1, \dots, t$, and from (2.83) and (2.84) we know that $|H \cap T_j| = |T_j \setminus H| = 1$. We will now inspect the three parts of the sum on the right-hand side of (2.91) for any given tooth $k = 1, \dots, t$.

1. From constraints (2.69) and (2.70) we have that

$$\begin{aligned} u(\delta(H \cap T_k)) = \sum_{d \in D} \sum_{j \in N} \left(u^d_{i_{1k}, j} + u^d_{j, i_{1k}} \right) = \\ 1 + (n^S - 1) \sum_{d \in D} z^d_{i_{1k}} + 1 - \sum_{d \in D} z^d_{i_{1k}} = \\ 2 + (n^S - 2) \sum_{d \in D} z_{i^d_{1k}}. \end{aligned}$$

2. Similarly as above, $u(\delta(T_k \setminus H)) = 2 + (n^S - 2) \sum_{d \in D} z^d_{i_{2k}}$.

3. From $|T_k| = 2$ and $T_K \subseteq N^0$, it follows that $r^S(T_k) \geq 1$, and from constraint (2.78) we have that $u(\delta(T_k)) = \sum_{d \in D} \sum_{i \in T_k} \sum_{j \notin T_k} \left(u^d_{ij} + u^d_{ji} \right) \geq 2r^S(T_k) \geq 2$. Note that we

can never assume that $r^S(V) > 1$ for any $V \subset N^0$, $|V| = 2$, as V may contain a starting point and even though no edges are entering the subset V, both nodes inside are visited as in Figure 2.4.

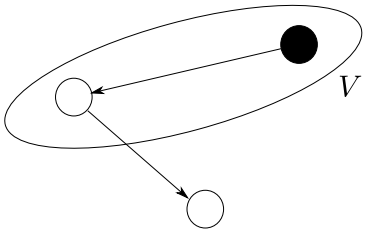


Figure 2.4: Minimum $r^S(V)$ when $|V| = 2$ and there exists $i \in V$ such that $z_i > 0$.

Plugging these into (2.91) we have that

$$u(\delta(H)) + \sum_{j=1}^{t} u(\delta(T_j)) \geq$$
$$\frac{1}{2}\left[\sum_{j=1}^{t}\left(2 + (n^S - 2)\sum_{d\in D} z^d_{i_{1j}}\right) + \sum_{j=1}^{t}\left(2 + (n^S - 2)\sum_{d\in D} z^d_{i_{2j}}\right) + \sum_{j=1}^{k} 2r^S(T_j)\right] \geq$$
$$\frac{1}{2}\left[2\cdot 2t + (n^S - 2)\sum_{j=1}^{t}\sum_{d\in D}\left(z^d_{i_{1j}} + z^d_{i_{2j}}\right) + 2t\right] \geq$$
$$3t + \frac{1}{2}(n^S - 2)\sum_{j=1}^{t}\sum_{d\in D}\left(z^d_{i_{1j}} + z^d_{i_{2j}}\right).$$

□

2.7.2 Generalised strengthened comb inequalities

Building on the generalised 2-matching inequalities from Section 2.7.1, we propose a generalisation of the comb inequalities. Using the definition of *power set*, $P(N) = \{V | V \subseteq N\}$, let us first define the following notation:

1. **Starting point counter.** We define the function $\zeta : P(N) \to \mathbb{N}$, representing *the number of starting points* present in the subset V over the entire time horizon.

$$\zeta(V) = \sum_{d\in D}\sum_{i\in V} z^d_i, \quad V \subseteq N. \tag{2.92}$$

2. **Set exiting counter.** The function $r^S : P(N) \to \mathbb{N}$ was previously defined as the *upper bounding function* in Section 2.4. However, here we are also considering subsets that contain the fixed depot:

$$r^S(V) = \left\lceil \frac{|V|}{Q^S_{max}} \right\rceil, \quad V \subset N,$$

where $Q^S_{max} = \left\lceil \frac{n^C}{n^D n^s} \right\rceil$ is the maximum route length. This function counts the edges that exit the co-boundary of the set V. For consistency we will preserve the

notation used by Letchford et al [91], $\tilde{r}^S(V) = r(N \setminus V)$ when $0 \in V$ and $\tilde{r}^S(V) = r(V)$ when $0 \notin V$.

3. **Set entering counter.** We define $\hat{r}^S : P(N) \to \mathbb{N}$ as

$$\hat{r}^S(V) = \max\left\{\left\lceil \frac{|V| - \zeta(V) n^S Q^S_{max}}{Q^S_{max}} \right\rceil, 0\right\} = \max\left\{\left\lceil \frac{|V|}{Q^S_{max}} \right\rceil - \zeta(V) n^S, 0\right\}, \quad (2.93)$$

where $\hat{r}^S(V)$ is the minimum number of sellers that will enter set $V \subseteq N$. Note that, if there is no starting point in set V, then $r^S(V) = \hat{r}^S(V)$. Analogously as in the previous point, $\tilde{\hat{r}}^S(V) = \hat{r}^S(N \setminus V)$ when $0 \in V$, and $\tilde{\hat{r}}^S(V) = \hat{r}^S(V)$ when $0 \notin V$.

4. **Comb exiting counter.** The function $S^S : P(N^{t+1}) \to \mathbb{N}$ counts the number of edges that exit each one of the sets in its domain:

$$S^S(H, T_1, \ldots, T_t) = \sum_{j=1}^{t} \left(\tilde{r}^S(T_j \cap H) + \tilde{r}^S(T_j \setminus H) + \tilde{r}^S(T_j)\right).$$

5. **Comb entering counter.** Similarly as above, the function $\sigma : P(N^{t+1}) \to \mathbb{N}$ counts the number of edges that enter each set in its domain:

$$\sigma(H, T_1, \ldots, T_t) = \sum_{j=1}^{t} \left(\tilde{\hat{r}}^S(T_j \cap H) + \tilde{\hat{r}}^S(T_j \setminus H) + \tilde{\hat{r}}^S(T_j)\right).$$

For a given comb with handle H and teeth $T_1, \ldots, T_t$ we define the *generalised strengthened comb inequality* as follows:

$$u(\delta(H)) + \sum_{j=1}^{t} u(\delta(T_j)) \geq \frac{1}{2}\left[S^S(H, T_1, \ldots, T_t) + \sigma(H, T_1, \ldots, T_t)\right]. \quad (2.94)$$

If there are no starting points present in the comb, then $\hat{r}(V) = r(V)$ for any part of the comb $V \in \{H, T_1 \ldots T_t\}$, and consequently $\sigma(H, T_1, \ldots, T_t) = S^S(H, T_1, \ldots, T_t)$. Then the generalised strengthened comb inequality is reduced to the strengthened comb inequality [91]. These strengthened comb inequalities, however, are valid for the truck problem by definition, as this problem fits the requirements of the classic CVRP:

$$x(\delta(H)) + \sum_{j=1}^{t} x(\delta(T_j)) \geq S^T(H, T_1, \ldots, T_t).$$

Here we use the standard notation $x(\delta(V)) = \sum_{d \in D} \sum_{i \in V, j \notin V} (x^d_{ij} + x^d_{ji})$. The comb entering counter is $S^T(H, T_1, \ldots, T_t) = \sum_{j=1}^{t} \left(\tilde{r}^T(T_j \cap H) + \tilde{r}^T(T_j \setminus H) + \tilde{r}^T(T_j)\right)$, where $\tilde{r}^T(V) = r^T(N \setminus V)$ when $0 \in V$, and $\tilde{r}^T(V) = r^T(V)$ when $0 \notin V$. Similarly as r^S, we adjust the definition of the upper bounding function from Section 2.5 to consider subsets that contain the fixed depot: $r^T(V) = \left\lceil \frac{|V|}{Q^T_{max}} \right\rceil$, where $V \subset N$.

Proposition 2.7.2. *The generalised strengthened comb inequalities are valid for the seller problem.*

Proof. Starting as in Theorem 2.7.1:

$$u(\delta(H)) + \sum_{j=1}^{t} u(\delta(T_j)) \geq \frac{1}{2}\left[\sum_{j=1}^{t} u(\delta(H \cap T_j)) + \sum_{j=1}^{t} u(\delta(T_j \backslash H)) + \sum_{j=1}^{t} u(\delta(T_j))\right],$$

we will decompose the right-hand side into three parts in the order shown above for any given $k = 1, \ldots, t$:

$$\begin{aligned} u(\delta(H \cap T_k)) &= \sum_{d \in D} \sum_{i \in H \cap T_k} \sum_{j \notin H \cap T_k} \left(u_{ij}^d + u_{ji}^d\right) \\ &= \sum_{d \in D} \sum_{i \in H \cap T_k} \sum_{j \notin H \cap T_k} u_{ij}^d + \sum_{d \in D} \sum_{i \in H \cap T_k} \sum_{j \notin H \cap T_k} u_{ji}^d. \end{aligned} \tag{2.95}$$

The first part in (2.95) represents the number of sellers leaving the boundary of the customer set $H \cap T_k$, that is:

$$\sum_{d \in D} \sum_{i \in H \cap T_k} \sum_{j \notin H \cap T_k} u_{ji}^d \geq \tilde{r}^S(H \cap T_k).$$

The second part in (2.95) is the number of sellers entering the boundary of $H \cap T_k$, :

$$\sum_{d \in D} \sum_{i \in H \cap T_k} \sum_{j \notin H \cap T_k} u_{ji}^d \geq \tilde{\tilde{r}}^S(H \cap T_k).$$

Then

$$u(\delta(H \cap T_k)) \geq \tilde{r}^S(H \cap T_k) + \tilde{\tilde{r}}^S(H \cap T_k).$$

As above, we obtain that:

$$\begin{aligned} u(\delta(T_k \backslash H)) &\geq \tilde{r}^S(T_k \backslash H) + \tilde{\tilde{r}}^S(T_k \backslash H), \\ u(\delta(T_k)) &\geq \tilde{r}^S(T_k) + \tilde{\tilde{r}}^S(T_k). \end{aligned}$$

Finally, we have that:

$$\begin{aligned} &u(\delta(H)) + \sum_{j=1}^{t} u(\delta(T_j)) \geq \\ &\frac{1}{2} \sum_{j=1}^{t} \left[\tilde{r}^S(H \cap T_j) + \tilde{\tilde{r}}^S(H \cap T_j) + \tilde{r}^S(T_j \backslash H) + \tilde{\tilde{r}}^S(T_j \backslash H) + \tilde{r}^S(T_j) + \tilde{\tilde{r}}^S(T_j)\right] = \\ &\frac{1}{2}\left[S^S(H, T_1, \ldots, T_t) + \sigma(H, T_1, \ldots, T_t)\right]. \end{aligned} \tag{2.96}$$

□

As a result of (2.93), inequality (2.94) is nonlinear. Since our aim is to keep the model linear, it is necessary to amend (2.93).

Corollary 1. *The linear equality* $\hat{r}^S(V) = \left\lceil \frac{|V|}{Q} \right\rceil - \mu_V n^S$ *preserves the validity of* (2.94) *for the Seller Problem.*

Proof. The linear equality $\hat{r}^S(V) = \left\lceil \frac{|V|}{Q} \right\rceil - \mu_V n^S$ is a valid lower bound of (2.93). The

proof follows directly as the left-hand side of (2.94) remains lower than the right-hand side. □

2.8 Exact method

In this section we propose an exact algorithm using the branch-and-cut methodology reviewed in Section 1.6 to solve the two-stage multi-period VRP with depot location and workload balance.

Our proposed algorithm is given in Algorithm 2.1 in the form of a recursive procedure. We are solving a relaxation of the problem from Section 2.6, defined with (2.68)– (2.76). During the branch-and-bound (B&B) algorithm, depending on the solution in each explored node of the tree, we run a different *separation procedure* to identify violated inequalities according to the following criteria:

- **Fractional solution:** search for violated generalised (or strengthened) combs.
- **Integer solution:** search for violated workload balance constraints.

Since in a B&B tree the number of fractional solutions reported is considerably higher than the number of integer feasible solutions, a separation procedure that is ran at these nodes can become overly memory-heavy and time consuming. In order to control this better, we introduce a *frequency parameter* ϕ, which is a whole number that appoints how many fractional solutions should be skipped before the separation procedure is run again.

The procedure is run recursively until the stopping criteria is met: either an optimal solution has been found, or the time has reached a certain time limit t_{max}. The separation procedures differ depending on whether their job is to identify violations in the seller or the truck routes (u or x decision variables in Section 2.6.3). These procedures will be detailed in the sections that follow.

2.8.1 Separation procedure for comb inequalities

In this section we outline a separation procedure for comb inequalities valid for the seller and truck problems. Many sophisticated heuristic separation procedures for the comb and two-matching inequalities have been developed in the past 50 years. In general, the approaches start by establishing a handle, and then "growing" teeth following some rules. An overview can be found in Section 1.6.

We propose an algorithm that differs from most common comb separation procedures in the sense that we first construct teeth, then connect them with a handle. We found that this approach is particularly useful when it comes to eliminating multiple subtours with a single inequality. The foundation of our algorithm lies on the *block structures* and *cut nodes* of the graph. Procedures established on these properties of the graph are known under the general name of *block heuristics*.

We will explain the separation procedure in the context of the seller problem to avoid confusion. The same procedure is applied to the truck problem, with the only

difference that the resulting valid inequality is the simpler, strengthened comb, as opposed to the generalised comb from Section 2.7.2 for the seller problem.

Algorithm 2.1 Branch-and-cut algorithm

Define problem $P \leftarrow$(2.68) – (2.76).
procedure SOLVERELAXATION(P)
  Initialise B&B tree with stopping criteria t_{max} and frequency parameter $\phi \in \mathbb{N}$.
  $Opt \leftarrow$ *false*, $t \leftarrow$ clock time
  Set $k \leftarrow$ root node, $ID \leftarrow 0$
  while $Opt = false$ AND $t < t_{max}$ **do**
    Set $u_k^* \leftarrow$ seller problem solution at B&B node k
    Set $x_k^* \leftarrow$ truck problem solution at B&B node k
    Set $Const_v \leftarrow \emptyset$ to hold violated constraints
    if u_k^* is fractional AND $ID \mod \phi = 0$ **then**
      $Const_v \leftarrow Const_v \cup$ COMBSEPARATION(u_k^*)
    else if u_k^* is integer **then**
      $Const_v \leftarrow Const_v \cup$ WBSELLERSEPARATION(u_k^*, z_k^*)
    if c_k^* is fractional AND $ID \mod \phi = 0$ **then**
      $Const_v \leftarrow Const_v \cup$ COMBSEPARATION(x_k^*)
    else if c_k^* is integer **then**
      $Const_v \leftarrow Const_v \cup$ WBTRUCKSEPARATION(x_k^*)
    if $Const_v = \emptyset$ AND (x_k^*, u_k^*, z_k^*) is optimal **then**
      $Opt \leftarrow true$
    else if $Const_v = \emptyset$ AND (x_k^*, u_k^*, z_k^*) is not optimal **then**
      Set $k \leftarrow$ next B&B node, $ID \leftarrow$node ID of k
    else if $Const_v \neq \emptyset$ **then**
      $P \leftarrow P \cup Const_v$
      SOLVERELAXATION(P) `/* Restart procedure with updated P`
  return best solution found

procedure WBSELLERSEPARATION(u^*, z^*)
  Identify violated workload balance constraints for the seller subproblem.

procedure WBTRUCKSEPARATION(x^*)
  Identify violated workload balance constraints for the truck subproblem.

procedure COMBSEPARATION(u^*)
  Identify violated comb inequalities.

The procedure operates on a *connected, undirected* graph defined as $G^{'}(N, E^{'}_{u^*})$ where N is the customer set, including the fixed depot, and

$$E^{'}_{u^*} = \{(i,j) \mid i,j \in N, \sum_{d \in D} (u_{ij}^{d*} + u_{ji}^{d*}) > 0\}$$

is the set of undirected edges with lengths extracted from the solution of the seller

problem. An edge from node i to node j is only defined if in the current solution a seller travels directly either from i to j or vice versa on any given day. At the start, the cut-nodes and blocks from the graph G' are identified and stored. This procedure is given in Appendix B. We define a parameter θ as an odd number greater than 2 that defines the desired comb size. It is interesting to eliminate subtours early at a fractional solution of the branch-and-bound tree. For this purpose we split the set of identified blocks such that we first consider the blocks that do not contain the depot. The complete procedure is outlined in Algorithm 2.2.

Algorithm 2.2 Generalised comb separation

procedure COMBSEPARATION(u^*)
  Initialise undirected graph $G(N, E_{u^*})$
  $T \leftarrow \{\emptyset\}$, $H \leftarrow \emptyset$, $C \leftarrow \{H, T\}$ /* Sets to store combs
  $B \leftarrow \emptyset, B_D \leftarrow \emptyset, B_{ND} \leftarrow \emptyset$ /* Sets of blocks
  $P_{cut} \leftarrow \emptyset$ /* Cut nodes
  $(P_{cut}, B) \leftarrow$ FINDBLOCKS($G(N, E_{u^*})$)
  $B_D \leftarrow \{b \mid b \in B,\ b \cap \{0\} \neq \emptyset\}$
  $B_{ND} \leftarrow \{b \mid b \in B,\ b \cap \{0\} = \emptyset\}$
  /* First explore all blocks not containing the depot, then the remaining
  for b in $\{B_{ND}, B_D\}$ **do**
    if $b \cap P_{cut} \neq \emptyset$ **then**/* There are cut-nodes in block b
      $T \leftarrow T \cup b$ /* Add tooth to set T
      $b_{min} \leftarrow \arg\min\{|b'| \mid b' \cap b \neq \emptyset\}$/* Smallest neighbour block
      /* b_{min} exists as G is connected
      $H \leftarrow H \cup b_{min}$/*Add nodes from b_{min} to handle
    else
      $T \leftarrow T \cup b$ /* Add tooth to set T
      $H \leftarrow H \cup \{i\}, i \in b$ random /* Add random tooth node to handle
    if $|T| = \theta$ **then**
      $C \leftarrow C \cup \{H, T\}$ /* Store comb
      $T \leftarrow \{\emptyset\}$, $H \leftarrow \emptyset$ /* Reset values
  if $|T| \geq 3$ and odd **then**/* Satisfies comb conditions
    $C \leftarrow C \cup \{H, T\}$
  else if $|T| > 3$ and even **then**/* Does not satisfy comb conditions
    remove one tooth from T
    $C \leftarrow C \cup \{H, T\}$
  return Generalised comb inequalities for all combs in C.

procedure FINDBLOCKS($G(N, E)$)
  Returns (P_{cut}, B), lists of cut nodes and all blocks in G

2.8.2 Separation procedure for workload balance inequalities

In the complete three-index formulation from Section 2.6.3, (2.77) – (2.80) contain an exponential number of constraints that control the workload balance by cardi-

nality of the sellers and the trucks. Our branch-and-cut algorithm (Algorithm 2.1) relies on an ad hoc insertion of these constraints by monitoring the branching tree and investigating each feasible solution. For this purpose an adequate separation procedure is necessary to identify the precise inequalities that are violated and add them to the initial model.

Different separation procedures are used for the seller and the truck problem. Constraints (2.77) and (2.79) are separated using Algorithm 2.3, whereas for constraints (2.78) and (2.80) we use Algorithm 2.4.

The foundation of these algorithms is the *shrinking heuristic* (Section 1.6). Taking the seller problem as an example, consider day d of the planning horizon and a given intermediate solution (u^{d*}) representing all of the sellers' tours on day d. For a given parameter $0 < \epsilon \leq 1$ we are considering all edges on the graph $G''(N, E''_{u^{d*}})$, where N is the set of all customers and the depot, and $E''_{u^{d*}}$ is

$$E''_{u^{d*}} = \{(i, j) \mid i, j \in N, u^{d*}_{ij} \geq \epsilon\}.$$

The graph G'' is pre-processed to eliminate the starting points in the case of the seller problem. If a node i is identified to be a starting point, all nodes that are connected to i become connected to the depot 0 instead, and their connection to the starting point is temporarily removed.

We are interested in identifying all *connected subgraphs* in G'' and forming *families of nodes*. For each identified edge (i, j) of length at least ϵ, we 'shrink' the edge into a single node $n_{new} = |N| + 1$. The new node n_{new} is in turn connected to the rest of the graph in the same way as i and j were. The old nodes i and j then become the *parents* of n_{new}, and a family is formed. The procedure continues and the shrinking of any edge whose end-node is n_{new} results in a new child-node, while n_{new} becomes a parent, and they all remain in the same family. Using this procedure, all connected subgraphs of G'' form their own families, and the original subgraph can be found by backtracking the family tree of the youngest child-node. Copies of the depot are created in the process to distinguish between different tours that contain the depot but are not related.

At the beginning, every original node is assigned a value of 1. The value of every child-node is calculated as a sum of its parents values. This way we keep track of the size of the connected subgraph, and effectively the *size of the path*. These values determine whether or not a given family violates a workload balance constraint. These procedures return a set of detected violated inequalities.

The procedure is given in Algorithm 2.3 for the truck problem and in Algorithm 2.4 for the seller problem. The procedure that forms families of nodes using connected subgraphs via shrinking is given in Algorithm 2.5.

Algorithm 2.3 Workload balance separation: truck problem

procedure WBTRUCKSEPARATION(x^*)
  Initialise $N_{parents} \leftarrow \{\emptyset\}$ /* Set of triples ($parent_1$,$parent_2$, n_{new})
  for d in D **do**
    $x^{d*} \leftarrow$ solution belonging to day d
    $N_{parents} \leftarrow$FORMFAMILIES(x^{d*})
    $Const_x \leftarrow \emptyset$
    for f in $N_{parents}$ **do** /* Loop through all families
      Identify all relatives of f_3, ignoring depot 0
      Store all relatives in V
      if V violates (2.77) **then**
        $Const_x \leftarrow Const_x \cup$ (2.77)
      else if V violates (2.79) **then**
        $Const_x \leftarrow Const_x \cup$ (2.79)
  return $Const_x$

procedure FORMFAMILIES(x^*)

Algorithm 2.4 Workload balance separation: seller problem

procedure WBSELLERSEPARATION(u^*, z^*)
  Initialise $N_{parents} \leftarrow \{\emptyset\}$ /* Set of triples ($parent_1$,$parent_2$, n_{new})
  for d in D **do**
    $u^{d*} \leftarrow$ solution belonging to day d
    for i in N **do**
      if $z_i^{d*} > 0$ **then**
        for $j \in N$, such that $u_{ij}^{d*} > 0$ **do** /* Alter u^{*d}
          $u_{j0}^{d*} \leftarrow u_{ji}^{d*}$
          $u_{ji}^{d*} \leftarrow 0$
    $N_{parents} \leftarrow$FORMFAMILIES(u^{d*}) /* ($parent_1$,$parent_2$, n_{new})
    $Const_u \leftarrow \emptyset$
    for f in $N_{parents}$ **do**
      Identify all relatives of f_3, ignoring depot 0
      Store all relatives in V
      if V violates (2.78) **then**
        $Const_u \leftarrow Const_u \cup$ (2.78)
      else if V violates (2.80) **then**
        $Const_u \leftarrow Const_u \cup$ (2.80)
  return $Const_u$

procedure FORMFAMILIES(u^*)

Algorithm 2.5 Edge shrinking procedure

procedure FORMFAMILIES(u^*)
  Initialise $n_{nodes} \leftarrow n^C + 1$ /* Track updated number of nodes
  $M \leftarrow u^*$ and set principal diagonal to 1
  $e_{sh} \leftarrow true$ /* Shrinkable edge indicator
  $N_{parents} \in \mathbb{N}^3, N_{parents} \leftarrow \emptyset$ /* Store parents of each new node
  while $e_{sh} = true$ **do**
    $e_{sh} \leftarrow false$
    for $i, j \in \{0, \dots n_{nodes}\}$ **do**
      if $M_{ij} > \epsilon$ AND $i \neq j$ **then**
        $e_{sh} \leftarrow true$
        $n_{nodes} \leftarrow n_{nodes} + 1$
        $N_{parents} \leftarrow N_{parents} \cup (i, j, n_{nodes})$
        $M \leftarrow$ SHRINKEDGE(M,I,J)
  return $N_{parents}$

procedure SHRINKEDGE(M, i, j)
  if $i = 0$ **then** Create copy of depot $0'$ not connected to j, set $i \leftarrow 0'$
  if $j = 0$ **then** Create copy of depot $0'$ not connected to i, set $j \leftarrow 0'$
  Shrink edge (i, j) into a single node n_{nodes}
  connect n_{nodes} to every node connected to i and j
  $M_{n_{nodes}, n_{nodes}} \leftarrow M_{ii} + M_{jj}$
  return M

2.9 Results

In this section we explore the quality of the solution methods presented above.

2.9.1 Data sets

The Two-stage multi-period VRP with depot location and workload balance is a problem inspired by a company based in Ecuador. The company has 1,756 customers located within the boundaries of a city. A river divides the customers (Figure 2.1), making it easier for a vehicle to visit two customers on the same side of the river, compared to a mix of customers located on both sides.

We have generated random subsets of the original customer location set. The subsets vary as follows:

- **Geographically.** The customers split into two groups to account for their location with respect to the river. We consider three classes: all customers west of the river (class R_w); all customers east of the river (class R_e); customers proportionally divided between the east and west side of the river as calculated from the original data set (class R_p).
- **Number of customers.** For each $n^C = \{20, 25, 30, 35, 40, 45\}$ and each geographical class, three data sets are randomly generated. There is a total of 54 different data

sets of customer coordinates.

- **Parameter values.** For every data set, 3 instances are generated for different values of the parameters n^S, n^T, n^D.

The total number of instances generated for testing is 162. The properties of the instances are summarised in Table 2.1.

n^C	Parameter range	R_w	R_e	R_p	Num of instances
20	{2,3}	3	3	3	27
25	{2,3}	3	3	3	27
30	{2,3}	3	3	3	27
35	{2,3}	3	3	3	27
40	{2, 3, 4}	3	3	3	27
45	{2, 3, 4}	3	3	3	27

Table 2.1: Properties of the testing instances

We denote this set of instances with I. The name of each instance $i \in I$ used in the analysis that follows is defined as n^C + *orientation* + *cardinal number*.

2.9.2 Exact methods

In this section, we analyse the two proposed models for the two-stage multi-period VRP with depot location and workload balance. These are the four-index model (Section 2.6.1) which can be handled by a solver alone, and the three-index model (Section 2.6.3) which is solved using the specialised branch-and-cut algorithm outlined in Section 2.8. All experiments are run on a Dell PowerEdge R740 server running Scientific Linux 7, with 4 Intel Gold 6234 processor with 8 cores, 16 threads and 1.5TB disk space. All optimization is performed using IMB CPLEX 12.8.0 through the Concert Technology library in C++.

Cutting strategy

We examine four different strategies of executing the two different separation procedures presented in Section 2.8.1 and Section 2.8.2 during the branch-and-cut algorithm.

S1 Generalised comb inequality separation procedure executed at the root node, and in every node with an ID number in the branching tree divisible by the product of the number of customers, sellers, trucks, and days, $n^C n^S n^T n^D$. The separation procedure for workload balance inequalities is ran in every integer node of the branching tree.

S2 Generalised comb inequality separation procedure executed at the root node, and in every node with an ID number in the branching tree divisible by the number of customers, n^C. The separation procedure for workload balance inequalities is executed in every integer node of the branching tree.

S3 Generalised comb inequality separation procedure executed in every node with an ID number in the branching tree smaller than the product of the number of customers, sellers, trucks, and days, $n^C n^S n^T n^D$. The separation procedure for workload balance inequalities is ran in every integer node of the branching tree.

S4 Generalised comb inequality separation procedure executed at the root node, and in every node with an ID number in the branching tree divisible by the product of the number of customers, sellers, trucks, and days, $n^C n^S n^T n^D$. The separation procedure for workload balance inequalities is ran in every other node of the branching tree tree, including all integer nodes. In the separation procedure from Section 2.8.2, we set $\epsilon = 0.5$.

S5 Only separation procedure for workload balance inequalities is executed in every integer node of the branching tree tree.

To decide on the best strategy for the branch-and-cut algorithm, we performed a series of experiments. Every strategy was tested on each of the 162 instances from Section 2.9.1. Each experiment had a time limit of 2 hours. We compare the scenarios to each other, analysing the following:

- The number of instances for which a given strategy resulted in the best *solution optimality gap* with respect to the rest of the strategies. Assume that after running instance $i \in I$ with strategy S_j, $j = \{1 \dots 5\}$, for 2 hours, the algorithm finds x^*_{ij} to be the best feasible solution, and $x^{'}_{ij}$ to be the best solution of the relaxed problem (Section 2.8.2). The solution gap is calculated as

$$gap_i^j = \frac{x^*_{ij} - x^{'}_{ij}}{x^*_{ij}}.$$

- The number of instances for which a given strategy resulted in a worse gap compared to the rest of the strategies.

- The average solution optimality gap ($\overline{gap}^j$) for each strategy S_j, $j = \{1, \dots, 5\}$, over all instances $i \in I$.

- The deviation for each strategy from the mean gap calculated between all strategies

$$dev_j = \sqrt{\sum_{i \in I} \frac{(gap_i^j - \mu_i)^2}{|I|}},$$

  where $j \in \{1, \dots 5\}$ is the index of a strategy. For every instance, $i \in I$ the mean instance gap is calculated as

$$\mu_i = \frac{\sum_{j=1}^{5} gap_i^j}{5}.$$

- Taking the information above into account, we assign a *score* to each strategy. We calculate this as the difference between the number of best gaps and the number

of worst gaps, and multiply this number by the inverse of the corresponding deviation (1-dev_j). If the number of instances where strategy j performs badly is high compared to the cases where it performs well, the score is negative. We are interested in the strategy with the *highest positive score*. This implies that there are more instances where this strategy showed the best performance, and the deviation from the overall average gap is relatively low. The latter is desirable as spikes in the optimality gap (e.g. instances with no upper bound identified) are minimised.

Strategy	S 1	S 2	S 3	S 4	S 5
Num. of best gaps	38	33	25	48	13
Num. of worst gaps	18	23	28	30	92
Average gap (%)	26.7	29.6	29.7	26.9	67.1
Deviation (%)	11.2	11.7	12.4	12.4	13.3
Score	17.8	8.8	−2.6	15.8	−68.5

Table 2.2: Analysis of solution gaps from 162 instances

Using the information in Table 2.2 we can choose S1 as the most desirable strategy.

In the following we will provide further analysis of the cutting strategy S1 by evaluating two scenarios:

1. Executing the workload balance separation procedure in fractional nodes.
2. Removing the valid comb inequalities from the algorithm.

It is intuitive to ask whether the workload balance inequalities, which are necessary for feasibility, should be identified and included more vigorously. This is strategy S4, where we run the corresponding separation procedure in fractional solutions as well in hopes that we will eliminate infeasible candidates earlier in the solution process. The parameter ϵ is set to 0.5, which means that the graph considered for separation only consists of edges with weight greater than 0.5.

In Figure 2.5 we compare the average optimality gap obtained after 2 hours with both S1 and S4, for all 162 instances grouped by size. As this result is inconclusive and one strategy does not dominate the other, we will proceed to explore other properties.

In Figure 2.6 we explore the relative change in upper bound and number of cuts identified after 2 hours with S1 and S4. The relative change of upper bounds is calculated as $(BUB^{S4} - BUB^{S1})/BUB^{S4}$, where BUB is the upper bound and the superscript specifies strategy used. Over all instances, S4 results in an upper bound that is 3% higher than the average upper bound obtained by S1. The relative change of the average number of cuts added to the feasible region is calculated as $(NC^{S4} - NC^{S1})/NC^{S4}$, where NC represents the number of cuts, and the superscript is the strategy. On average, S4 includes 53% more inequalities in the feasible region in the form of cuts compared to S1. Both changes suggest dominance of S1 over S4. Even though the number of additional cuts added to the problem with S4 is significantly higher compared to S1, the solution quality is worsened. The increase of inequalities and the accelerated execution of the workload balance separated procedure has

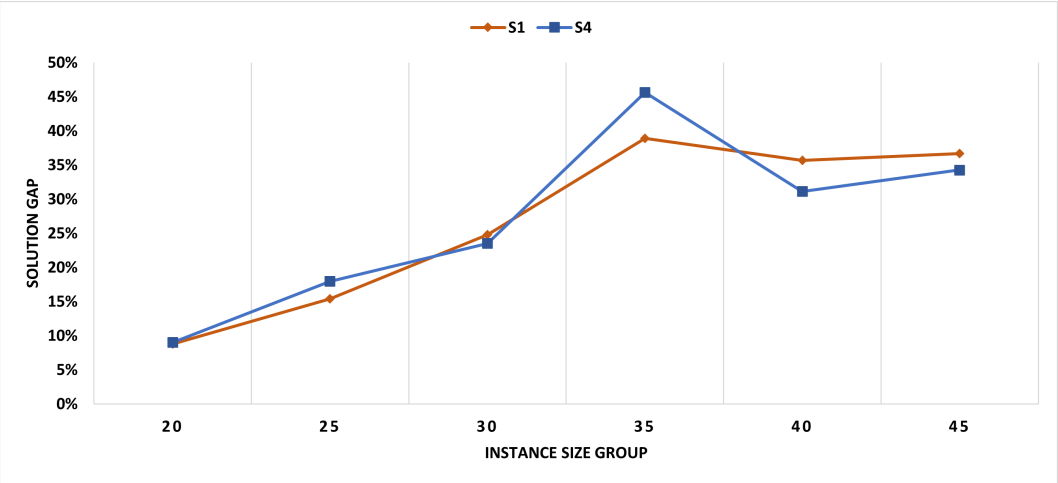


Figure 2.5: Average optimality gap for S1 and S4

an impact on the memory that the algorithm consumes, resulting in a more costly solution process, while the solution quality is not improved.

We conclude that strategy S1 dominates S4, and it is not beneficial to separate workload balance constraints more often than necessary.

For the second point of interest, we consider strategy S5, where the valid comb inequalities are eliminated from the algorithm. In Figure 2.7 we compare the average optimality gap obtained after 2 hours with both S1 and S5, for all 162 instances grouped by size. It is clear that S1 dominates S5 in every size group, and we will take this as conclusive evidence supporting our initial claim.

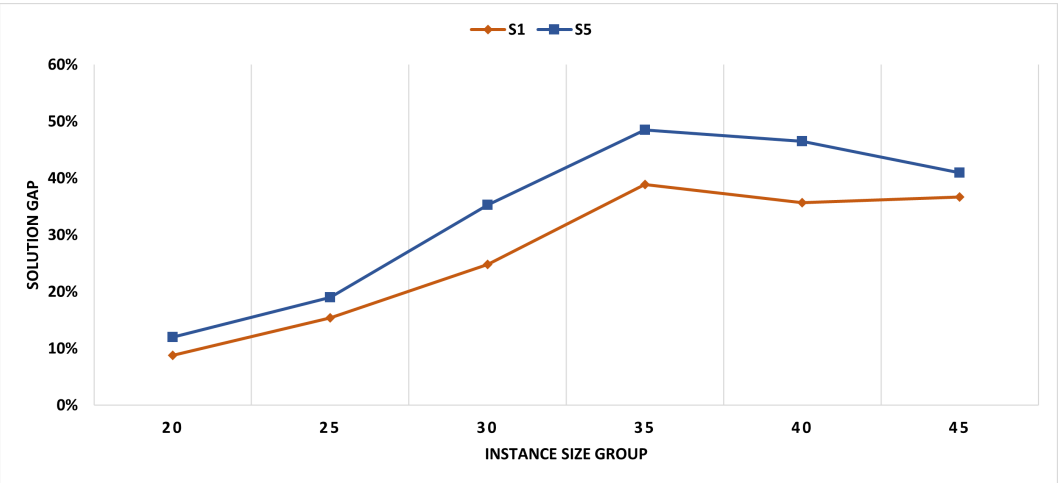
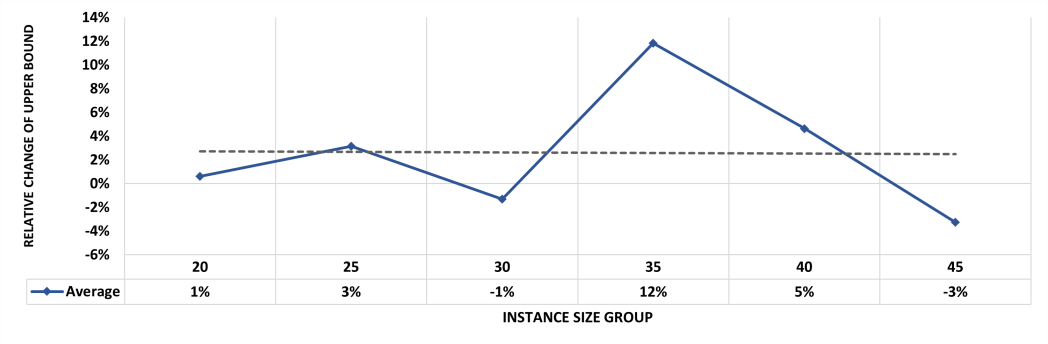


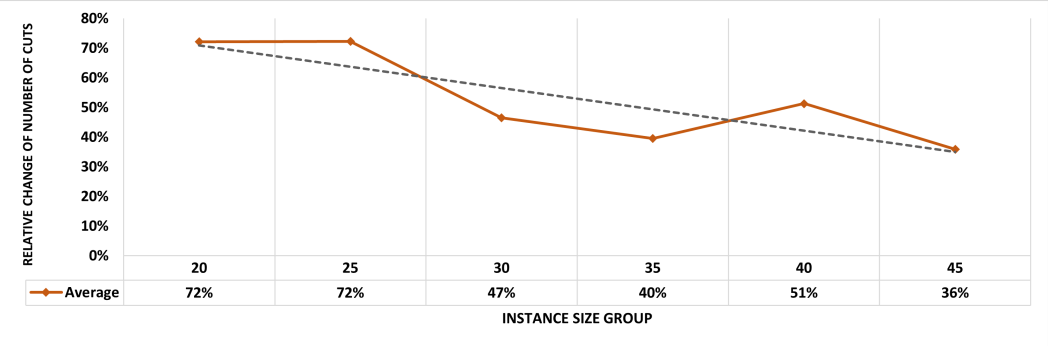
Figure 2.7: Average optimality gap for S1 and S5

Performance analysis

The performance of the 3-index model solved with the branch-and-cut algorithm and S1 will be measured against the 4-index model solved by CPLEX. Every one of the 162 instances from Section 2.9.1 is ran with a time limit of 2 hours and the results are divided by instance size in Table 2.3 – Table 2.8.



(a) Relative change of average upper bound for S1 and S4 per instance size



(b) Relative change of average number of cuts for S1 and S4 per instance size

Figure 2.6: Additional analysis of scenarios S1 and S4

	Parameters				3-index model				4-index model		
Data set	n^C	n^T	n^S	n^D	**BUB**	**Gap**	**NC**	**T(s)**	**BUB**	**Gap**	**T(s)**
20W1	20	2	2	3	9236	9.49%	513	7200	9202	32.14%	7200
20W1	20	2	2	2	**7470**	**0%**	917	782	7470	30.69%	7200
20W1	20	2	3	2	8423	7%	1153	7200	8409	30.39%	7200
20W2	20	2	2	3	9421	10.11%	668	7200	9295	27.16%	7200
20W2	20	2	2	2	7576	2.53%	1432	7200	7566	23.75%	7200
20W2	20	2	3	2	8406	9.40%	1430	7200	8398	27.84%	7200
20W3	20	2	2	3	12912	23.96%	1408	7200	12372	36.19%	7200
20W3	20	2	2	2	9306	13%	3274	7200	9289	32.08%	7200
20W3	20	2	3	2	11258	24.74%	3052	7200	10959	35.99%	7200
20E1	20	2	2	3	6462	8.74%	942	7200	6783	25.77%	7200
20E1	20	2	2	2	5437	7%	2725	7200	5402	21.96%	7200
20E1	20	2	3	2	6097	8.25%	1757	7200	6021	17.79%	7200
20E2	20	2	2	3	7685	24.78%	943	7200	7016	35.48%	7200
20E2	20	2	2	2	5573	10.67%	1271	7200	5573	32.61%	7200
20E2	20	2	3	2	6309	14.02%	1714	7200	6508	37.60%	7200
20E3	20	2	2	3	6065	12.09%	662	7200	6122	22.54%	7200
20E3	20	2	2	2	5054	3.28%	1484	7200	5054	21.48%	7200
20E3	20	2	3	2	5567	6.22%	1208	7200	5621	22.25%	7200
20P1	20	2	2	3	14282	3.92%	619	7200	14715	23.84%	7200
20P1	20	2	2	2	**12990**	**0%**	1027	3445	13026	26.63%	7200
20P1	20	2	3	2	14480	2.33%	881	7200	14443	27.02%	7200
20P2	20	2	2	3	15928	10.45%	532	7200	15887	36.64%	7200
20P2	20	2	2	2	13556	3%	1099	7200	13566	24.85%	7200
20P2	20	2	3	2	15146	10.37%	1074	7200	16166	44.15%	7200
20P3	20	2	2	3	14826	9.30%	450	7200	15273	32.31%	7200

20P3	20	2	2	2	**12613**	**0%**	1274	5099	12613	33.17%	7200
20P3	20	2	3	2	14041	1.86%	696	7200	14026	33.47%	7200
	Average				9856	8.78%	1267	6745	9881	29.47%	7200

Table 2.3: Solutions for 20-customer instances

	Parameters				**3-index model**				**4-index model**		
Data set	n^C	n^T	n^S	n^D	**BUB**	**Gap**	**NC**	**T(s)**	**BUB**	**Gap**	**T(s)**
25W1	25	2	2	3	13217	37.75%	2270	7200	12086	49.89%	7200
25W1	25	2	2	2	9854	24.64%	6990	7200	8854	38.75%	7200
25W1	25	2	3	2	11386	32.36%	6284	7200	10081	43.76%	7200
25W2	25	2	2	3	7636	8.14%	918	7200	7643	25.96%	7200
25W2	25	2	2	2	6561	3.17%	1188	7200	6561	23.12%	7200
25W2	25	2	3	2	7208	6.76%	2350	7200	7298	26.41%	7200
25W3	25	2	2	3	9748	28.78%	1495	7200	10447	53.66%	7200
25W3	25	2	2	2	8291	20.94%	3171	7200	7703	42.40%	7200
25W3	25	2	3	2	9082	18.80%	2813	7200	8181	44.82%	7200
25E1	25	2	2	3	6974	24.01%	1912	7200	6692	29.37%	7200
25E1	25	2	2	2	5415	8.39%	1802	7200	5293	17.70%	7200
25E1	25	2	3	2	6261	19.04%	3520	7200	5786	21.55%	7200
25E2	25	2	2	3	6040	20.13%	1598	7200	5944	30.54%	7200
25E2	25	2	2	2	4744	7.95%	2833	7200	4925	23.90%	7200
25E2	25	2	3	2	5138	8.55%	1755	7200	5470	23.28%	7200
25E3	25	2	2	3	6184	20.50%	971	7200	6458	38.85%	7200
25E3	25	2	2	2	4856	9.32%	3009	7200	4935	25.29%	7200
25E3	25	2	3	2	5021	6.28%	1480	7200	5071	22.80%	7200
25P1	25	2	2	3	18457	28.12%	2329	7200	16878	45.86%	7200
25P1	25	2	2	2	12378	1.80%	939	7200	12396	35.17%	7200
25P1	25	2	3	2	13808	4.28%	1380	7200	13869	38.73%	7200
25P2	25	2	2	3	21606	34.22%	1897	7200	18250	45.71%	7200
25P2	25	2	2	2	12996	1.69%	1944	7200	13034	33.46%	7200
25P2	25	2	3	2	14648	5.84%	1674	7200	14876	36.49%	7200
25P3	25	2	2	3	17611	27.93%	2452	7200	16799	51.16%	7200
25P3	25	2	2	2	12230	2.44%	2205	7200	12259	43.73%	7200
25P3	25	2	3	2	13650	3.48%	1665	7200	13879	47.13%	7200
	Average				10037	15.38%	2328	7200	9691	35.54%	7200

Table 2.4: Solutions for 25-customer instances

	Parameters				**3-index model**				**4-index model**		
Data set	n^C	n^T	n^S	n^D	**BUB**	**Gap**	**NC**	**T(s)**	**BUB**	**Gap**	**T(s)**
30W1	30	2	2	3	14800	42.28%	4540	7200	12631	60.87%	7200
30W1	30	2	2	2	10953	37.34%	8975	7200	8686	47.17%	7200
30W1	30	2	3	2	12711	43.30%	9591	7200	11209	57.09%	7200
30W2	30	2	2	3	12349	38%	4404	7200	9530	33.57%	7200
30W2	30	2	2	2	6921	2.84%	3223	7200	7089	23.92%	7200
30W2	30	2	3	2	7663	7.78%	3502	7200	7714	23.29%	7200
30W3	30	2	2	3	/	100%	3436	7200	11916	56.80%	7200
30W3	30	2	2	2	8241	16.61%	3418	7200	8974	44.02%	7200
30W3	30	2	3	2	10840	34%	6670	7200	10022	49.45%	7200
30E1	30	2	2	3	10676	47.89%	3989	7200	8948	50.38%	7200
30E1	30	2	2	2	7226	24.05%	7599	7200	6537	42.38%	7200

30E1	30	2	3	2	8161	33.85%	9188	7200	7067	41.63%	7200
30E2	30	2	2	3	7580	26%	4864	7200	6951	41.92%	7200
30E2	30	2	2	2	4898	5.23%	2369	7200	5258	29.52%	7200
30E2	30	2	3	2	5597	9.09%	3199	7200	5830	31.67%	7200
30E3	30	2	2	3	9322	43%	5137	7200	6539	41.41%	7200
30E3	30	2	2	2	4900	7.72%	3165	7200	5267	31.71%	7200
30E3	30	2	3	2	5509	10%	2440	7200	5918	36.46%	7200
30P1	30	2	2	3	19212	30%	5352	7200	17811	52.57%	7200
30P1	30	2	2	2	**12002**	**0%**	473	39	15106	49.02%	7200
30P1	30	2	3	2	**13428**	**0%**	1642	2313	14380	43.49%	7200
30P2	30	2	2	3	28494	40%	4472	7200	23639	54.38%	7200
30P2	30	2	2	2	20154	28.11%	8594	7200	18954	49.34%	7200
30P2	30	2	3	2	19770	20%	3359	7200	21963	52.36%	7200
30P3	30	2	2	3	19259	19%	4210	7200	18577	50.14%	7200
30P3	30	2	2	2	**12553**	**0%**	2473	6030	13748	39.70%	7200
30P3	30	2	3	2	14232	3.68%	1000	7200	16126	45.54%	7200
Average					11825	24.81%	4492	6710	11348	43.70%	7200

Table 2.5: Solutions for 30-customer instances

	Parameters				**3-index model**				**4-index model**		
Data set	n^C	n^T	n^S	n^D	**BUB**	**Gap**	**NC**	**T(s)**	**BUB**	**Gap**	**T(s)**
35W1	35	2	2	3	13561	33.79%	3771	7200	/	100.00%	7200
35W1	35	2	2	2	12871	37%	12058	7200	12413	45.81%	7200
35W1	35	2	3	2	12398	29.56%	7581	7200	11859	42%	7200
35W2	35	2	2	3	/	100%	4610	7200	/	100%	7200
35W2	35	2	2	2	9957	25%	9508	7200	7999	17.78%	7200
35W2	35	2	3	2	8772	12%	9076	7200	/	100%	7200
35W3	35	2	2	3	/	100%	4429	7200	/	100%	7200
35W3	35	2	2	2	9978	13.45%	8647	7200	10562	32.18%	7200
35W3	35	2	3	2	12181	25.91%	7831	7200	/	100%	7200
35E1	35	2	2	3	10185	39.65%	5610	7200	/	100%	7200
35E1	35	2	2	2	7835	30%	11543	7200	7088	40.09%	7200
35E1	35	2	3	2	9311	36.14%	11270	7200	7481	39.40%	7200
35E2	35	2	2	3	8552	34.55%	4079	7200	/	100%	7200
35E2	35	2	2	2	5682	11%	5043	7200	6101	30.71%	7200
35E2	35	2	3	2	7515	32.77%	12435	7200	/	100%	7200
35E3	35	2	2	3	/	100%	4655	7200	9384	55.03%	7200
35E3	35	2	2	2	8241	38%	11582	7200	7240	46.38%	7200
35E3	35	2	3	2	9724	43.94%	12523	7200	8835	53%	7200
35P1	35	2	2	3	29502	43.93%	4681	7200	26920	68.72%	7200
35P1	35	2	2	2	20425	38%	12103	7200	21462	66.36%	7200
35P1	35	2	3	2	22106	34.29%	12514	7200	22410	65%	7200
35P2	35	2	2	3	/	100%	4259	7200	/	100%	7200
35P2	35	2	2	2	17578	30%	11555	7200	16366	52.84%	7200
35P2	35	2	3	2	17163	22.47%	11720	7200	18322	57%	7200
35P3	35	2	2	3	22033	30.69%	4641	7200	/	100%	7200
35P3	35	2	2	2	13651	5%	5153	7200	16404	51.22%	7200
35P3	35	2	3	2	15188	7.02%	2648	7200	/	100%	7200
Average					13235	39.05%	7982	7200	13178	69.02%	7200

Table 2.6: Solutions for 35-customer instances

Parameters					3-index model				4-index model		
Data set	n^C	n^T	n^S	n^D	**BUB**	**Gap**	**NC**	**T(s)**	**BUB**	**Gap**	**T(s)**
40W1	40	2	2	2	13845	40%	13423	7200	11727	34.71%	7200
40W1	40	2	4	2	13291	32%	10227	7200	13656	37.51%	7200
40W1	40	4	2	2	13824	19%	6264	7200	16829	45.36%	7200
40W2	40	2	2	2	13075	31%	7119	7200	12866	42.92%	7200
40W2	40	2	4	2	/	100%	14299	7200	15008	45.91%	7200
40W2	40	4	2	2	17671	36%	8926	7200	15466	43.84%	7200
40W3	40	2	2	2	14174	40%	13513	7200	10938	35.34%	7200
40W3	40	2	4	2	/	100%	8671	7200	16959	52.72%	7200
40W3	40	4	2	2	17246	37%	9328	7200	14777	44.39%	7200
40E1	40	2	2	2	5539	4%	3993	7200	6802	29.65%	7200
40E1	40	2	4	2	8577	33%	12139	7200	8836	39.73%	7200
40E1	40	4	2	2	10199	34%	9573	7200	8614	31.17%	7200
40E2	40	2	2	2	8282	40%	12911	7200	6087	34.38%	7200
40E2	40	2	4	2	10098	47%	13502	7200	8897	51.34%	7200
40E2	40	4	2	2	9928	37%	8258	7200	8813	47.04%	7200
40E3	40	2	2	2	5672	16%	8449	7200	6020	37.70%	7200
40E3	40	2	4	2	9085	43%	15630	7200	7982	51.05%	7200
40E3	40	4	2	2	7004	19%	6319	7200	6904	41.67%	7200
40P1	40	2	2	2	19653	36%	13097	7200	14743	46.85%	7200
40P1	40	2	4	2	21308	30%	12949	7200	19037	55.64%	7200
40P1	40	4	2	2	25406	32%	8045	7200	23477	60.23%	7200
40P2	40	2	2	2	19325	32%	14229	7200	15286	40.66%	7200
40P2	40	2	4	2	22951	34%	15720	7200	20659	52.49%	7200
40P2	40	4	2	2	23431	29%	8158	7200	22944	56.37%	7200
40P3	40	2	2	2	17616	24%	13707	7200	15678	42.07%	7200
40P3	40	2	4	2	17958	11%	5628	7200	25158	59.30%	7200
40P3	40	4	2	2	24892	27%	9747	7200	22133	50.28%	7200
Average					14802	35.71%	10512	7200	13937	44.83%	7200

Table 2.7: Solutions for 40-customer instances

Parameters					3-index model				4-index model		
Data set	n^C	n^T	n^S	n^D	**BUB**	**Gap**	**NC**	**T(s)**	**BUB**	**Gap**	**T(s)**
45W1	45	2	2	2	12248	40.04%	11928	7200	10573	48.95%	7200
45W1	45	2	4	2	14667	44.35%	14085	7200	13915	58.16%	7200
45W1	45	4	2	2	14956	40.82%	8257	7200	13783	55.93%	7200
45W2	45	2	2	2	14933	40.30%	11372	7200	13572	49.26%	7200
45W2	45	2	4	2	18938	49.01%	13749	7200	/	100%	7200
45W2	45	4	2	2	18829	41.60%	9377	7200	/	100%	7200
45W3	45	2	2	2	10462	24.94%	8177	7200	9185	27.29%	7200
45W3	45	2	4	2	14538	41.02%	13762	7200	13222	43.33%	7200
45W3	45	4	2	2	14415	33.32%	8703	7200	15311	49.37%	7200
45E1	45	2	2	2	8022	40.53%	15392	7200	6573	44.53%	7200
45E1	45	2	4	2	8088	38.61%	15210	7200	8237	53.63%	7200
45E1	45	4	2	2	8773	37.53%	10595	7200	7450	48.04%	7200
45E2	45	2	2	2	6661	18.53%	9377	7200	7166	44.85%	7200
45E2	45	2	4	2	8892	33.14%	14354	7200	10081	57%	7200
45E2	45	4	2	2	7722	16.23%	3811	7200	10094	56.05%	7200
45E3	45	2	2	2	9554	44.41%	14382	7200	7376	40.93%	7200
45E3	45	2	4	2	8859	34.63%	10891	7200	9944	52.75%	7200
45E3	45	4	2	2	11466	45.93%	10813	7200	10508	53.84%	7200
45P1	45	2	2	2	21268	36.72%	14320	7200	20132	52.67%	7200

45P1	45	2	4	2	27385	47.27%	14882	7200	24776	57.95%	7200
45P1	45	4	2	2	21783	25.01%	6571	7200	22585	52%	7200
45P2	45	2	2	2	20849	36.24%	11836	7200	19008	52.80%	7200
45P2	45	2	4	2	27653	45.41%	13895	7200	27808	66%	7200
45P2	45	4	2	2	23656	33.49%	7377	7200	28130	65.03%	7200
45P3	45	2	2	2	18258	25.68%	13728	7200	17655	50.42%	7200
45P3	45	2	4	2	20833	28.12%	13815	7200	25806	65.71%	7200
45P3	45	4	2	2	25707	36.71%	14950	7200	24117	62.42%	7200
	Average				15534	36.28%	11689	7200	15080	55.92%	7200

Table 2.8: Solutions for 45-customer instances

From the results above we observe that, on average, in 5 of the 6 instance sizes the 4-index model results in a better upper bound than the 3-index model. These results are visualised in Figure 2.8, where the horizontal axis shows ranges of relative change between the upper bound obtained with the 3-index model and the 4-index model.

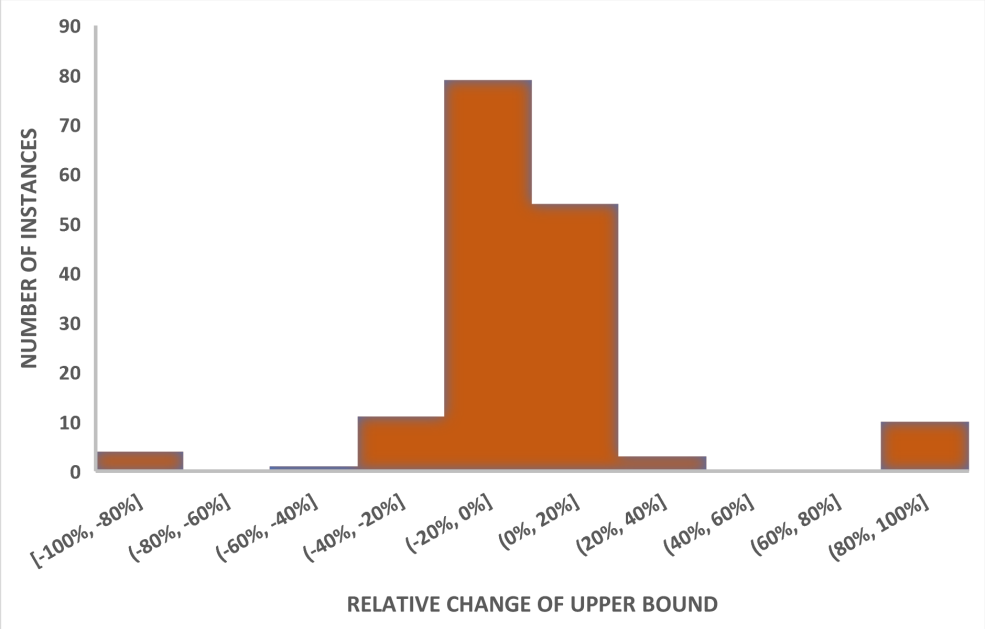


Figure 2.8: Upper bound analysis of 3-index vs 4-index model

However, in all of the instance sizes the average optimality gap is significantly smaller with the latter approach. The branch-and-cut procedure manages to better close the gap in 92% of the total number of instances, with an average improvement of 64%. The relative change is given in Figure 2.9.

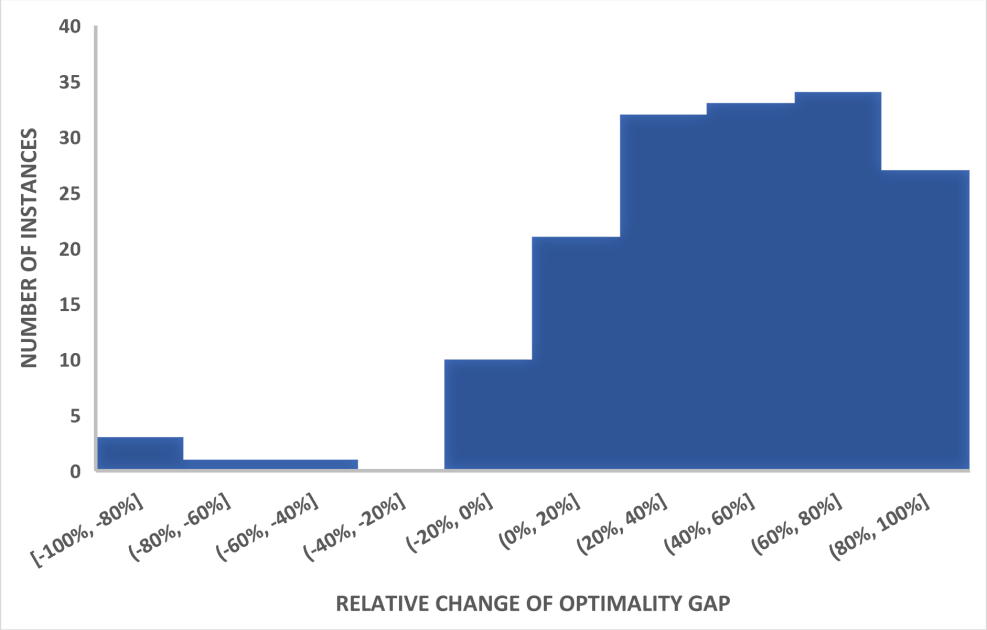


Figure 2.9: Optimality gap analysis of 3-index vs 4-index model

Another interesting feature of the two solution approaches that is worth analysing is the cost of executing the solution method for the imposed time limit of 2 hours. This can be measured in several ways, considering either the memory consumed by the algorithm, the number of nodes produced by the branching tree, the number of times an LP is solved during the course of the algorithm, etc. We opted for analysing the total number of simplex iterations performed in each of the two algorithms. The maximum number of iterations that can be performed is set to 9,223,372 trillion for both algorithms. The procedure will stop when the optimal solution is found, after two hours, or when this number of iterations is reached – whichever comes first. A higher number of iterations requires higher computational power.

In Figure 2.10 we analyse the averages of iterations for each of the instances within the six size groups. Three lines representing the minimum, maximum and average number of iterations (in millions) are shown for each of the two algorithms. From the graph, it is clear that even the minimum number of iterations performed for the 4-index model is higher than the maximum number of iterations performed for the 3-index model, for every group of instances. The average relative change in the number of iterations is 83%. This means that solving the 4-index model is significantly more expensive compared to the 3-index model and requires much higher computational power.

Another method for comparing the performance of different algorithms was proposed by Dolan and Moré [39]. This method creates performance profiles that easily compare the performance using a given metric. Using these performance profiles a comparison between the 3-index and 4-index models has been graphed in Figure 2.11. When using the upper bound as a performance metric as in the first graph, it is clear that the 4-index model outperforms the 3-index model. However, when the performance metric is the optimality gap, the 3-index model significantly outperforms the 4-index model, and this difference becomes even more significant

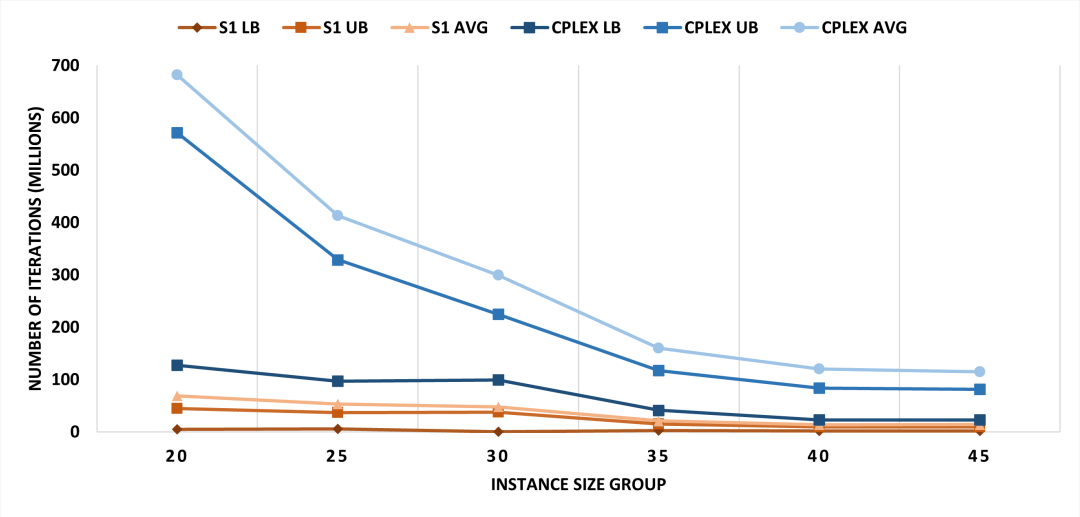


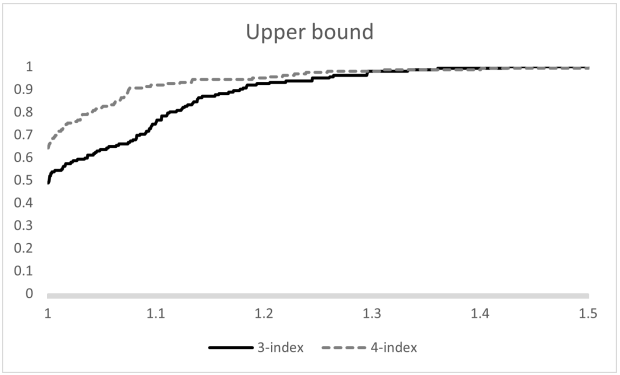
Figure 2.10: Number of BB iterations analysis of 3-index vs 4-index model

when the memory consumed during the course of the run-time is taken as a metric. We can conclude that overall, the 3-index model with the branch-and-cup algorithm outlined above is superior to the 4-index model solved by an MIP solver overall.

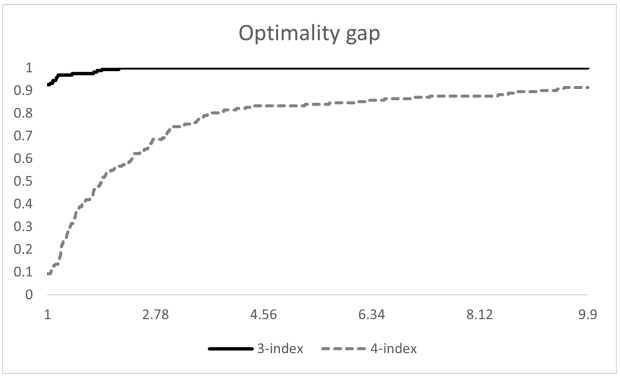
2.10 Conclusions

We introduced a complex routing problem motivated by the distribution operation with a network of 1,756 customers of a company based in Ecuador. The problem involves a set of trucks and sellers to be routed individually, location of starting points, a planning horizon and workload balancing. We proposed two models, and we performed an extensive computational analysis for both. Using the original data set provided by the company in question, we generated 162 instances ranging between 20 and 45 customers. We first introduce an extension of the 3-index VRP formulation, as this allows for use of adequate compact sets lower and upper bounding constraints. For this reason, theoretically it can be solved using a MIP solver alone. However, the 2-index VRP model is enough to describe a problem with a homogeneous fleet, like in our case, although the number of capacity constraints needed is exponential, which is why alternative solution methods need to be explored. Our problem is first formulated with the former approach and solved with a solver alone to serve as a benchmark. The second model uses the latter approach for which we developed a brand-and-cut algorithm.

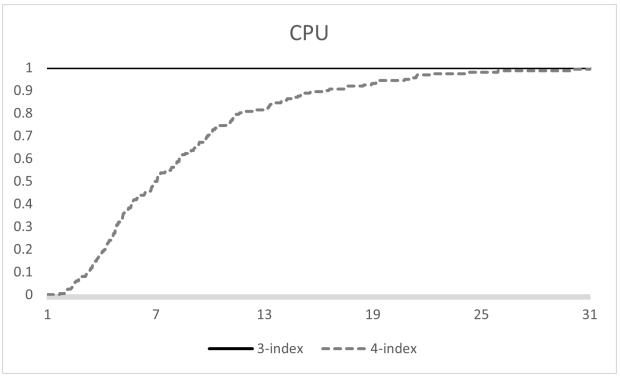
For the 3-index model we propose a variation of the workload balance constraints and a generalisation of the comb inequalities. Separation procedures for these are implemented within the branch-and-cut algorithm. We analyse a set of scenarios where different cuts are added at different times in the branching tree, and choose the best performing scenario. The results show that our proposed algorithm outperforms the solver alone as it results in a better optimality gap in 94% of the tested instances with an average improvement of 64%. The cost of running the algorithm is significantly lower, as the number of simplex iterations performed is 83% lower compared to the



(a) Upper bound obtained with 3 and 4 index models



(b) Optimality gap obtained with 3 and 4 index models



(c) Memory consumed obtained with 3 and 4 index models

Figure 2.11: Performance profiles for the 3 and 4 index models

algorithm used by CPLEX. On the other hand, CPLEX results in a better upper bound with an average difference of 3%. The computational experiments performed for the analysis of the exact method involved ~1,944 hours of computational time.

Chapter 3

Two-stage multi-period vehicle routing problem with depot location and workload balance: Heuristics

Exact methods have a limited ability when it comes to VRPs and similar problems. While plenty of excellent work has been done by the research community during the past few decades and impressive results have been achieved in this field, the industry needs require faster and more efficient solution methods. This resulted in a rich literature on different heuristic approaches. An overview is given in Section 1.7.

In this chapter we introduce a heuristic method for the two-stage multi-period VRP with depot location and workload balance, introduced in Chapter 2. This heuristic is based on the tabu-search methodology with a machine learning clustering method as a foundation. In Section 3.1 we propose a powerful construction heuristic that by itself produces promising results. Some improvement methods are given in Section 3.2, and the complete tabu-search framework is discussed in Section 3.3.

3.1 Construction heuristic

We propose a cluster-first route-second construction heuristic for the complete two-stage multi-period VRP with depot location and workload balancing in Section (2.1). The clustering is performed with the *k-means* method proposed by [94]. A survey can be found in [20]. The standard k-means procedure produces k clusters centred around some fixed points called *centroids*. The centroids dynamically change their location during the procedure, and in each iteration they are readjusted and placed in the weighted centre of the cluster. The coordinates of the centroids are calculated as averages of the coordinates of all points in the cluster. A given point belongs to cluster $i \leq k$ if the *squared distance* to the centroid c_i is smaller than its distance to any other centroid c_j, $j \leq k$, $j \neq i$.

We propose three mutations, each focusing on a different aspect of this problem:

1. **Converging clusters.** This procedure results in clusters that converge towards the mutual depot. The procedure is based on altering the classical k-means method to contain *multiple centroids*, and we call this procedure a *multi-centroid k-means*.

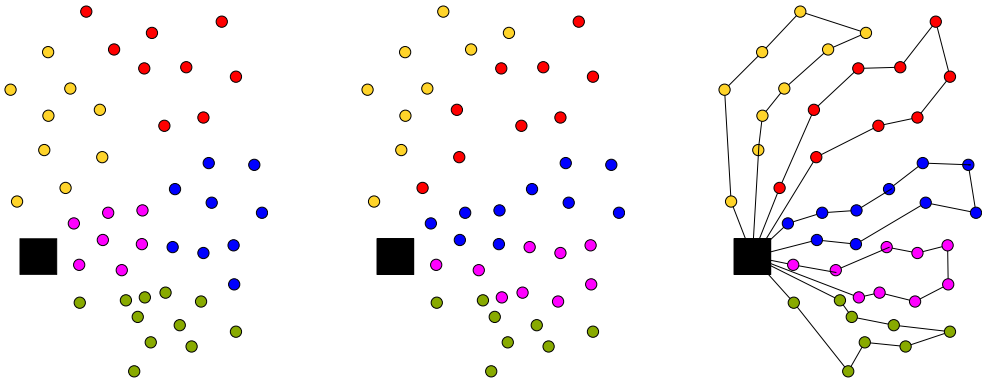
2. **Balanced clusters.** Considering that the workload balance by cardinality is crucial for this problem, we aim for balanced clusters with equal number of points when possible. This is achieved via the *balanced k-means*.

3. **Conjoint clusters.** The seller problem requires a second mutual point, other than the fixed depot, for all clusters. For this purpose, we introduce the *joint k-means*, where all clusters share an unknown point.

In the following, we elaborate on these three clustering procedures.

3.1.1 Multi-centroid k-means

The aim is to establish a clustering procedure which results in groups of customers that can be reasonably visited by the same vehicle starting and finishing at a fixed depot. An optimal VRP solution is unlikely to allocate a single vehicle to a group of customers centred around a single point far from the depot. Instead, an intuitive assumption is that a vehicle will visit customers along its way to its furthest point, as well as along its way back, as allowed by its capacity. Rather than clusters centred around a single point, it is more reasonable to form clusters that *converge towards the depot*. This logic is illustrated with a small example for $k = 5$ in Figure 3.1.



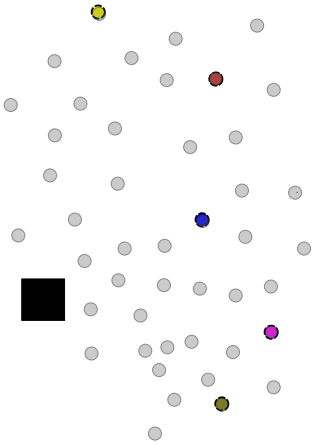
(a) Standard clusters. (b) Converging clusters. (c) VRP solution.

Figure 3.1: Adjusting customer clustering to routing problems with a fixed depot.

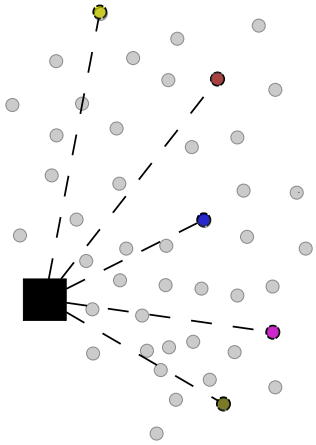
We create the converging clusters by assigning t centroids, $t \in \mathbb{N}$, $t \geq 2$ to each cluster, equally distributed and/or converging towards the depot in a straight line. This procedure is illustrated on the example above for $t = 5$ in Figure 3.2. To each centroid c_i^j, $1 \leq i \leq k$ and $2 \leq j \leq t$ we associate a *weight* $\lambda_i^j \in [0,1]$ that determines the strength by which the centroid j attracts points to its cluster i. If these weights are set to 1, each centroid is an equal contributor towards the creation of the clusters. If some centroids are set to less than 1, they will attract fewer surrounding points. For a given customer node p with plane coordinates (x_p, y_p) we determine the cluster to which it belongs by finding its closest centroid c_i^j with coordinates $(x_{c_i^j}, y_{c_i^j})$:

$$\arg\min_{\substack{1\leq i\leq k\\ 1\leq j\leq t}} \left(\left(x_p - x_{c_i^j}\right)^2 + \left(y_p - y_{c_i^j}\right)^2\right)\lambda_i^j. \tag{3.1}$$

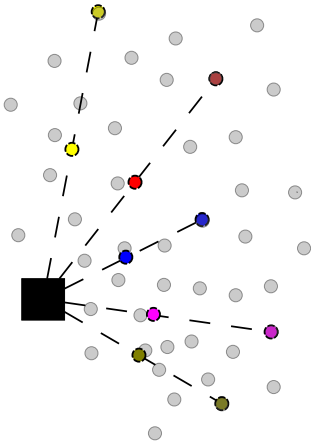
The complete procedure is given in Algorithm 3.1 on page 60.



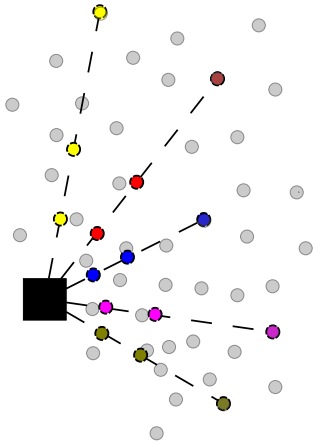
(a) Random assignment of first centroid.



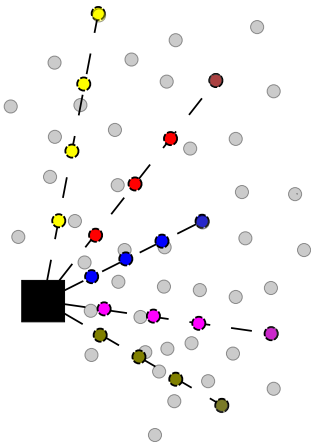
(b) Line connections between first centroids and depot.



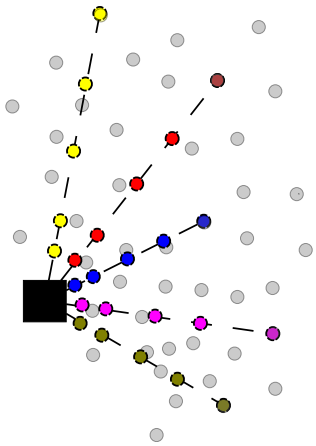
(c) Second centroid: middle point between first centroid and depot.



(d) Third centroid: middle point between depot and second centroid.



(e) Fourth centroid: middle point between first and second centroid.



(f) Fifth centroid: middle point between depot and third centroid.

Figure 3.2: Multiple centroid allocation illustration for 5 clusters and 5 centroids.

Algorithm 3.1 Multi-centroid k-means

procedure MULTICENTROIDCLUSTERING(k, t, N, λ) /* Assigns customers to clusters
  Initialise $C \leftarrow$CENTROIDALLOCATION(k, t, N)
  $clust : N^0 \rightarrow \{1,\ldots,k\}$ /* Customer to cluster allocation function
  for $p \in N^0$ **do**
    $clust_p \leftarrow$WEIGHTEDSQUAREDISTANCE(p, C, λ)
  return $clust$

procedure CENTROIDALLOCATION(k, t, N) /* Allocates t centroids in each of the k clusters
  for $i \in \{1,\ldots,k\}$ **do**
    $c_i^1 \leftarrow$ random $p \in N^0$
    $c_i^0 \leftarrow depot$
    $C_i \leftarrow (c_i^0, c_i^1)$ /* Distributed centroids from depot to furthest centroid
    $C_i^* \leftarrow (c_i^0, c_i^1)$ /* Updated set of distributed centroids
  for $i \in \{1,\ldots,k\}$ **do**
    $j \leftarrow 1$
    while $j \leq t$ **do**
      for $l \in C$ **do**
        if $l+1$ exists **then**
          $x_{c_i^j} \leftarrow$ MIDDLE(C_l, C_{l+1})
          INSERT c_i^j between elements l and $l+1$ in C_i^*
      $C \leftarrow C^*$ /* Update set of distributed centroids to include all new centroids
      $j \leftarrow j+1$
  $C \leftarrow C_1 \cup \cdots \cup C_k$
  return C

procedure MIDDLE(p_1, p_2) /* Finds middle point
  $x_{p_{mid}} \leftarrow (x_{p_1} + x_{p_2})/2$
  $y_{p_{mid}} \leftarrow (y_{p_1} + y_{p_2})/2$
  return p_{mid}

procedure WEIGHTEDSQUAREDISTANCE(p, C, λ) /* Finds weighted square distance
  $(i^*, j^*) \leftarrow \arg\min_{\substack{1\le i\le k \\ 2\le j\le t}}((x_p - x_{c_i^j})^2 + (y_p - y_{c_i^j})^2)\lambda_i^j$
  return i^*

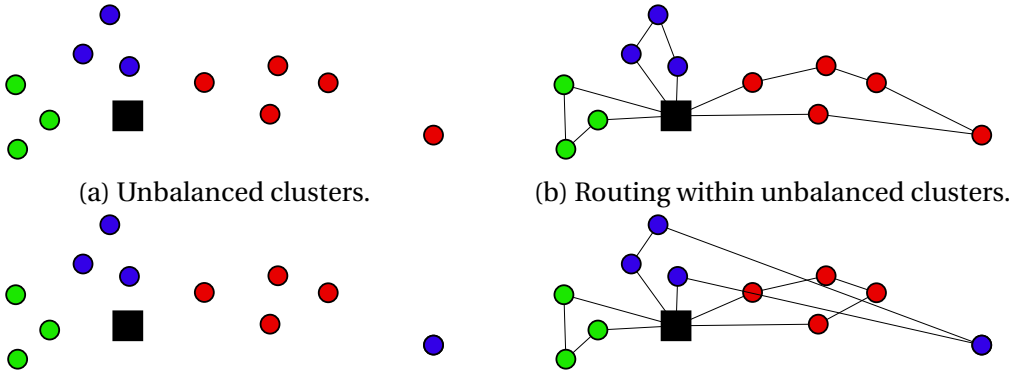
3.1.2 Balanced k-means

To construct feasible clusters of customers for all the vehicles, the cluster sizes have to remain within the bounds imposed by the workload balance constraints. Forcing

cluster sizes is counter-intuitive when it comes to the k-means procedure, as points are grouped only by their distance to the centroids.

Here we propose a procedure that will limit the cluster sizes, guaranteeing balanced cardinality. The procedure iteratively rearranges the points across the clusters in an informed manner, maintaining the cluster-like structure.

The intuitive reassignment is for every cluster to keep its nearest Q_{max} points and lose the rest. This means removing points that are along the edges of a cluster, far from its centres, and assigning them to another cluster nearby. However, this reasoning could lead to solutions that break cluster-like structure, significantly increasing the final cost of a routing problem. This case is shown in Figure 3.3.



(a) Unbalanced clusters.

(b) Routing within unbalanced clusters.

(c) Balanced clusters.

(d) Routing within balanced clusters.

Figure 3.3: Balancing clusters by keeping nearest points.

Instead of focusing on points to keep in a cluster, we focus on choosing points to lose. This means considering the alternative options for each point, and only losing the points that have the best second choice. This reasoning maintains both the balance and structure of all clusters.

Initially, all clusters are sorted by size and marked as *available* for new points. For every point we determine the nearest (current) and second nearest clusters as a minimum distance between the point a centroid in the given cluster. For a given point p, we denote these distances as d^1 and d^2 respectively. We define a *moving cost* $\alpha = d^2 - d^1$ as the added cost of assigning point p to its second nearest cluster. We know that the number of clusters with Q_{max} points is $UB = |N| - kQ_{min}$, and the number of clusters with Q_{min} points is $k - (|N| - kQ_{min})$. As the clusters are sorted, the first in the queue has the largest number of points - at least Q_{max}. We sort the points inside the cluster by increasing α - the first points in line have the lowest moving cost. We move all points after point Q_{max} (or Q_{min}, depending on the cluster's position in line) to their second nearest cluster, and mark the current cluster as *unavailable* for new members. We repeat the procedure for all clusters. The complete algorithm is given in Algorithm 3.2.

3.1.3 Joint k-means

During the process of creating clusters for the sellers on a given day d, it is particularly important to choose a starting point as this will be shared by all seller clusters.

Algorithm 3.2 Balanced k-means

procedure CLUSTERBALANCING(N, Q_{max},Q_{min},$clust$)
  /* Rearrange clusters to satisfy balance
  Initialise $C = \emptyset$, $K = \{\emptyset\}$
  $UB \leftarrow |N| - kQ_{min}$ /* Number of clusters with Q_{max}
  $LB \leftarrow k - UB$/* Number of clusters with Q_{min} points
  for $p \in N$ **do**
    /* Initialise distance to first and second nearest cluster
    $clust'_p \leftarrow$second nearest *available* cluster after $clust_p$
    $d_p^1 \leftarrow$ distance from p to nearest centroid in $clust_p$
    $d_p^2 \leftarrow$ distance from p to nearest centroid in $clust'_p$
  for $i = 1, \dots k$ **do**
    $C \leftarrow$ all $p \in N$ s.t. $clust_p = i$
    MARK C as *available*
    $K \leftarrow K \cup C$ /* Initialise cluster sets
  SORT list $K = \{C_1, \dots, C_k\}$ largest to smallest
  for $i \in 1, \dots, k$ **do**
    for $j \in C_i$ **do** $\alpha_{p_j} \leftarrow clust'_{p_j} - clust_{p_j}$
    SORT points in C_i by increasing α
    if $|C_i| > Q_{max}$ AND $i < UB$ AND $j > Q_{max}$ **then**
      $clust_{p_j} \leftarrow clust'_p$ /* Assign p_j to next available cluster
    else if $|C_i| > Q_{min}$ AND $i > LB$ AND $j > Q_{min}$ **then**
      $clust_{p_j} \leftarrow clust'_{p_j}$ /* Assign p_j to next available cluster
    MARK C_i as *unavailable*
  return $clust$ /* Return updated cluster assignment

The goal is to find a *common centroid*, other than the depot, which is located in a convenient location easily accessible to every seller. As this point will be a part of every seller tour on day d, it should be chosen strategically as to minimise the proximity to the first customer visited by every seller.

We propose a procedure of identifying a point that at the same time is as far as possible from the depot, and similarly distanced to the *main centroid of every seller cluster* on day d. Let $sp \in N$ be a customer location candidate for a starting point on day d. For every pair of main centroids, c_i and c_j, $1 \leq i \neq j \leq k$, we consider the absolute value of the difference between their distances to sp. The sum of these values will indicate the proximity of sp to all main centroids. Consider a special case where all main centroids are along the boundary of a circle. If then sp is placed in the centre of the circle, this sum will be 0. Therefore, we aim for a value that is as small as possible while pushing sp further from the depot:

$$\min_{sp \in N} \left\{ \sum_{0 \leq i \leq k} \sum_{\substack{0 \leq j \leq k \\ i \neq j}} \left| d(c_i^1, sp) - d(c_j^1, sp) \right| \right\}, \qquad \max_{sp \in N} d(sp, 0).$$

3.1.4 Complete clustering procedure

The complete clustering procedure is as follows:

Step 1 Create n^D converging, balanced clusters on the complete set of nodes N. These are the day clusters.

Step 2 Inside each day cluster, create n^T converging, balanced clusters. These are the truck clusters.

Step 3 Inside each day cluster, create n^S converging, balanced, conjoined clusters. These are the seller clusters.

Steps 2 and 3 can be executed simultaneously, or in any particular order. The proposed procedures for finding balanced and converging clusters can be applied to a variety of different problems, while the method for finding a common starting point is specialised to the Seller Problem.

The final remaining step of the construction heuristic is *routing*. As we have already established the exact sets of customers that each truck and each seller will visit on any given day, every cluster will consist of a single route. The problem is now reduced to $n^D\left(n^S + n^T\right)$ TSPs.

3.1.5 Routing

At this stage of the heuristic, the aim is to use a *fast* procedure that will construct a TSP route in each of the clusters defined above. Several simple and efficient construction heuristic procedures can be found in the literature that are commonly used for the TSP (Section 1.7). Among the best performing and fastest construction TSP heuristics, according to the literature and our computational study, is the rule of *random cheapest insertion*.

This procedure starts from a 2-node route, called a *seed*, and expands to cover all remaining nodes. In the truck clusters, the 2-node route consists of the depot and its furthest point. For the seller cluster we have modified the procedure to construct a *path* instead, starting from a single edge. The seed here is the edge between the depot and the established starting point.

Once the seeds are defined, the expanding procedure is the same for both the sellers and the trucks. The next node to join the route is chosen randomly and inserted into the route in the cheapest manner possible – where the added cost of inclusion is minimal. The insertion procedure is illustrated with a small example for a single truck route in Figure 3.4. The result for the trucks is a cycle, while for the sellers it is an open path, with the starting point and the depot as end-nodes.

3.2 Improvement methods

The construction heuristic outlined in Section 3.1 alone shows promising results. The performance of the algorithm will be discussed in the next section. To improve the solution further, we propose three improvement procedures that will later be combined into a large metaheuristic framework. Section 1.7 gives an overview of similar improvement heuristics.

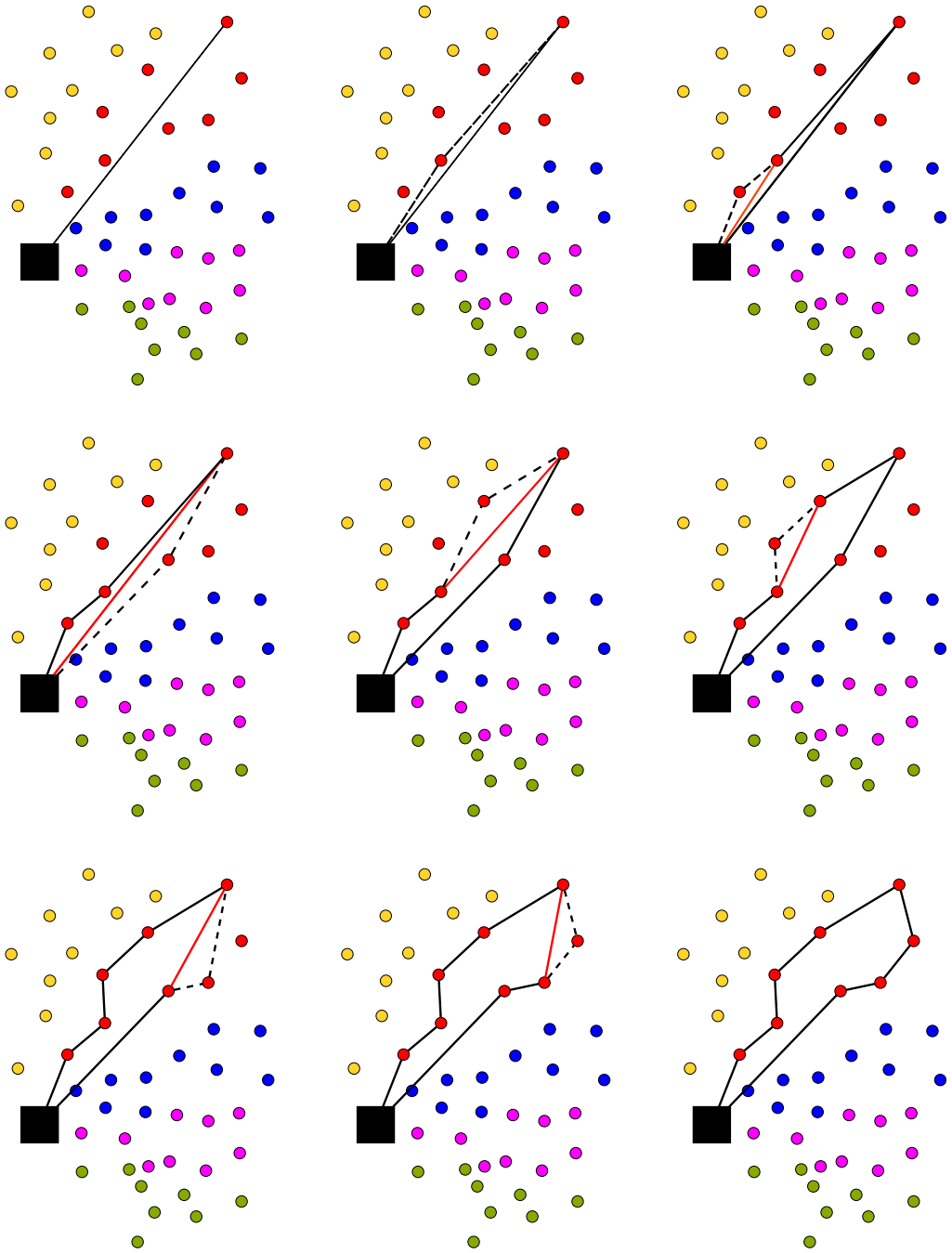


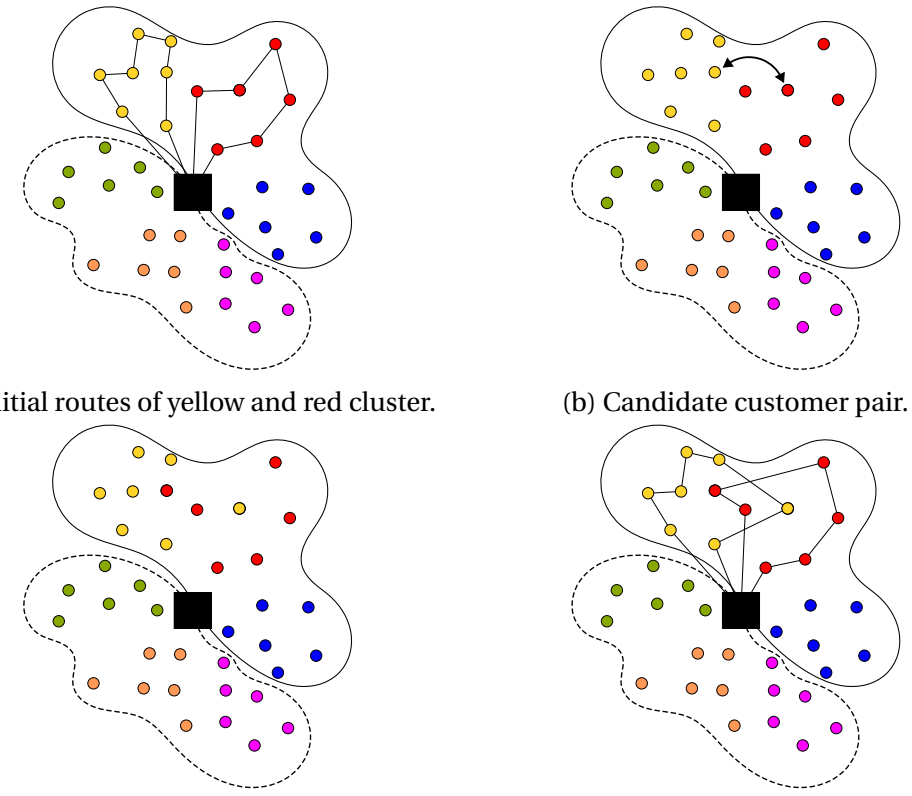
Figure 3.4: Random cheapest insertion TSP routing in a truck cluster.

3.2.1 Inter-route improvement

The main improvement efforts in this algorithm are based on the idea of *customer swaps*. The procedure is similar whether the operation in question concerns the sellers or the trucks. In the case of the sellers, however, the starting point cannot be a candidate for swapping. Once two trucks (or sellers) exchange one customer, their clusters preserve their original size but contain one "foreign" customer each. The routing then must be re-applied and the change in cost is closely monitored. If the new routes result in a lower overall cost, the change is labelled as “best". This process

is repeated for every pair of trucks (sellers) and every candidate pair of customers. In each iteration, the “best" swap is updated if an improvement is found. At the end of the procedure, this swap is applied to the solution. A small example of an inter-route improvement iteration is given in Figure 3.5. The example consists of two day clusters illustrated with a solid and a dashed curve. Each of the 6 colours represents a truck cluster.

We define a *tolerance factor* $\varepsilon \in [0,1]$ as the factor by which a worse solution is considered acceptable. If $\varepsilon = 0.01$, then solutions lower than the best solution + 1% are accepted.



(a) Initial routes of yellow and red cluster.

(b) Candidate customer pair.

(c) Inter-route customer swap.

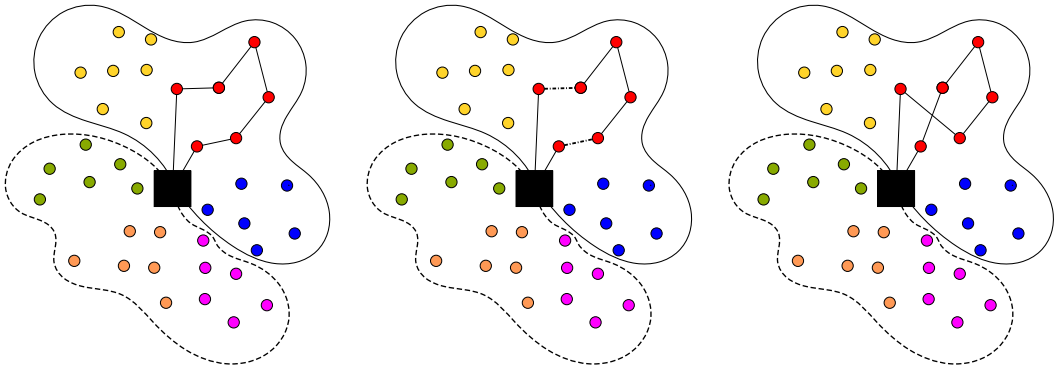
(d) New routes of yellow an red clusters.

Figure 3.5: Inter-route improvement of truck clusters.

3.2.2 Intra-route improvement

The routing performed within every truck and every seller cluster described in Section 3.1 is a TSP heuristic procedure. These routes can be further improved by applying some *TSP improvement heuristics*. A fast, simple, and efficient heuristic is the *2-opt* heuristic, a special case of the λ*-opt* algorithm introduced in [88]. This improvement procedure is a key ingredient of some of the best-performing solution methods, including the Lin-Kernighan heuristic [89]. The idea is to identify two non-neighbouring edges from a given route $(i, i+1)$ and $(j, j+1)$, where $i+1 < j$. For the seller routes we additionally impose that $i \neq sp$ and $j, j+1 \neq sp$, where 0 is the depot and sp is the starting point. These edges are then temporarily removed and replaced by (i, j) and $(i+1, j+1)$. If the routes are directed, the path $i+1, \ldots, j$ is reversed.

This procedure is repeated until either no more candidate pairs are left to explore, or some other stopping criteria is met (number of iterations, time etc). The change which results in the lowest routing cost becomes permanent. A single iteration of this procedure for a truck route is demonstrated on the same small example in Figure 3.6.



(a) Initial routes of red cluster. (b) Candidate edge pair. (c) 2-opt move.

Figure 3.6: Intra-route improvement of truck clusters.

3.2.3 Inter-day improvement

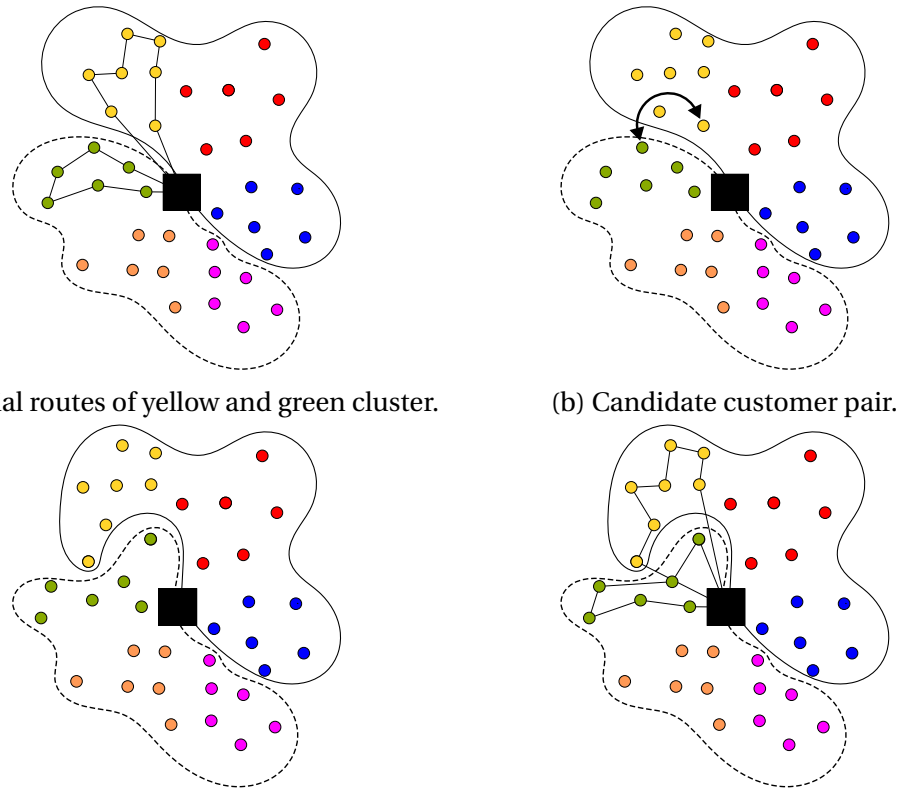
Like the inter-route improvements, the inter-day improvement procedure swaps two customers i and j from different day clusters. This results in four routes being effectively altered – the truck and seller routes where customers i and j originally belonged. After customer i (j) is removed its seller and truck clusters and replaced by j (i), the routing procedure from Section 3.1 is performed only on the changed clusters. The swap that grants the biggest cost improvement becomes permanent. A single iteration is illustrated in Figure 3.7. The two truck clusters in (a) belong to different days, and a candidate customer pair identified in (b) and swapped and the routing is repeated (c and d).

3.3 Tabu-search framework

In this section, we present a complete heuristic algorithm tailored for the two-stage multi-period VRP with depot location and route balancing. It is an adaptation of one of the best-performing VRP metaheuristics – Tabu Search. The algorithm consists of the following phases:

Phase 1 **Construction.** The construction phase is the cluster-first route-second procedure from Section 3.1.

- *Input parameters.* This phase takes the complete set of nodes N (customers and depot), number of days n^D, number of sellers n^S and number of trucks n^T. Additionally define the number of centroids t_r for each cluster and the vector of centroid weights λ_r, where $r = \{D, S, T\}$ representing days, sellers and trucks respectively. Once a non-improving solution is found, the solution is rejected and the clustering procedure for r terminates.



(a) Initial routes of yellow and green cluster.

(b) Candidate customer pair.

(c) Inter-day customer swap.

(d) New routes of yellow an green clusters.

Figure 3.7: Inter-day improvement of truck clusters.

Step 1: *Day clustering.* If $n^D > 1$, construct n^D converging, balanced clusters on the complete set of nodes N. The converging point is the depot, there are t_D centroids with weights λ_{t_D} and the stopping criteria is the discovery of a non-improving solution.

Step 2: *Parallel seller and truck clustering.* Executed in parallel, or in any arbitrary order

- If $n^T > 1$, construct n^T converging, balanced clusters on each of the n_D clusters from *Step 1.* The converging point is the depot and there are t_T centroids.
- If $n^S > 1$, construct n^S converging, balanced and joint clusters on each of the n_D clusters from *Step 1.* The converging point is the depot, there are $t_S + 1$ centroids, and the last centroid number $t_S + 1$ is a customer location shared among all n_S clusters.
- The centroids have assigned weights of λ_S (λ_T) and the stopping criteria is the discovery of a non-improving solution.

The output is a feasible solution S.

Phase 2 **Tabu search.** The tabu search phase consists of controlled exploration of the *neighbourhood of* S, $N(S)$ through *short-term memory*, a level of *tolerance*, solution *intensification* on the most *stable* routes, and solution *diversification*.

- *Input parameters.* This phase takes the feasible solution S from *Phase 1* along with some new control parameters. The memory-related parameter τ controls the capacity of this feature or, more precisely, the length of the tabu list. We define a stability parameter μ which limits the number of times a given route can be altered in order to qualify for intensification. We initiate an alteration counter $\mu_{s,t,d} = 0$ for every seller, truck and day cluster. Worse solutions are accepted with tolerance ε and with limit $\hat{\varepsilon}$.
- *Neighbourhood.* We define the neighbourhood of S, $N(S)$, as the set of all candidate solutions identified within a single execution of the *intra-route intensification procedure* from Section 3.2.
- *Short-term memory.* Once a solution $S' \in N(S)$ is accepted, it cannot be directly reversed back to S for as long as it is in the algorithm's short-term memory. This memory is the *tabu-list* of the algorithm, which deems the reverse moves tabu. The distinguishing properties of the move are added to the tabu-list as the newest entry e' corresponding to the transition to $S \rightarrow S'$. As the solution S' is obtained from S by swapping two customers from different route clusters within the same day cluster, these properties are: identification of route type (seller or truck), day cluster being altered, the seller (truck) in question and customer ID for both customers being swapped.
- *Tolerance.* Denote the cost of the best feasible solution $S' \in N(S)$ by $c(S')$. If $c(S') < c(S)$, the solution is accepted. Assume $c(S') \geq c(S)$. If $c(S') < c(S)(1+\varepsilon)$, and the number of solutions $S" \in N(S)$ accepted so far in this phase, such that $c(S) < c(S") < c(S)(1+\varepsilon)$, is lower than $\hat{\varepsilon}$, the solution S' is accepted.

Step 1: *Neighbourhood exploration.* Identify the best candidate $S' \in N(S)$, and go to *Step 2*.

Step 2: *Solution acceptance.* If the solution S' is not within the defined tolerance, or if the entry e' is already in the tabu-list, the solution is rejected. Go to *Step 3*. Otherwise, if there are already τ entries in the tabu-list, the oldest entry is "forgotten" and the current entry e' is aggregated to the list. If there are fewer than τ entries, the current entry e' is aggregated. For the seller/truck clusters between which the alteration is performed (members in e'), the corresponding parameter $\mu_{s,t,d}$ is increased by 1. The best solution is updated $S = S'$. Go to *Step 4*.

Step 3: *Stopping criteria.* If no non-tabu solution within the tolerance is identified, or the number of iterations exceeds the limit imposed t_{max}, pass the best solution found S to *Phase 3*. Otherwise, go to *Step 1*.

Step 4: ***Intensification.*** A second neighbourhood $N'(S)$ is defined on solution S as a set of all candidate solutions after a single execution of the intra-route improvement procedure from Section 3.2 for which $\mu_{s,t,d} < \mu$. Find the best candidate $S' \in N'(S)$ (if one exists), update $S = S'$ and return to *Step 1*.

Phase 3 **Diversification** This phase consists of solution diversification through the definition of a different general neighbourhood and new tolerance for acceptance.

- *Input parameters.* This phase takes the feasible solution S from *Phase 2*. Worse solutions are accepted with tolerance ε_d and with limit $\hat{\varepsilon}_d$.
- *Neighbourhood.* A new neighbourhood $N"(S)$ is defined on solution S as the set of all candidate solutions after a single execution of the inter-day improvement heuristic from Section 3.2.
- *Tolerance.* The solution tolerance is defined similarly as in *Phase 2*, with the new neighbourhood $N"$ and tolerance parameters ε_d and $\hat{\varepsilon}_d$.

Step 1: *Neighbourhood exploration.* Identify the best candidate $S' \in N"(S)$, and move to *Step 2*.

Step 2: *Solution acceptance.* If the solution S' is not within the defined tolerance, the solution is rejected and the algorithm goes to *Step 3*. Otherwise, update $S = S'$ and go to *Step 3*.

Step 3: *Stopping criteria.* If a solution within the tolerance is identified and the number of iterations is under the limit imposed t_{max}, pass the best solution found S back to *Phase 2*. Otherwise, Stop.

The list of input parameters used in this algorithm is given in the list below, followed by an illustration in Figure 3.8.

Input parameters:

λ_D : Weights for centroids in day clusters.
λ_T : Weights for centroids in truck clusters.
λ_S : Weights for centroids in seller clusters.
t_D : Number of converging centroids in each day cluster.
t_T : Number of converging centroids in each truck cluster.
t_S : Number of converging centroids in each seller cluster.
μ : Route alteration limit for intra-route improvements.
ε : Solution quality acceptance tolerance for inter-route improvements.
$\hat{\varepsilon}$: Limit of worse solutions acceptable for inter-route improvements.
ε_d : Solution quality acceptance tolerance for inter-day improvements.
$\hat{\varepsilon}_d$: Limit of worse solutions acceptable for inter-day improvements.
τ : Tabu list length.
t_{max} : Execution time limit.

3.4 Results

In this section we will analyse the performance of the heuristic algorithm outlined in Section 3.3 on all instances from Section 2.9.1. The solution quality will be quantified as the relative difference from the best solution obtained with an exact algorithm, either with the 3-index or the 4-index model. We will take a closer look at the construction heuristic (Section 3.1) before exploring the tabu-search framework.

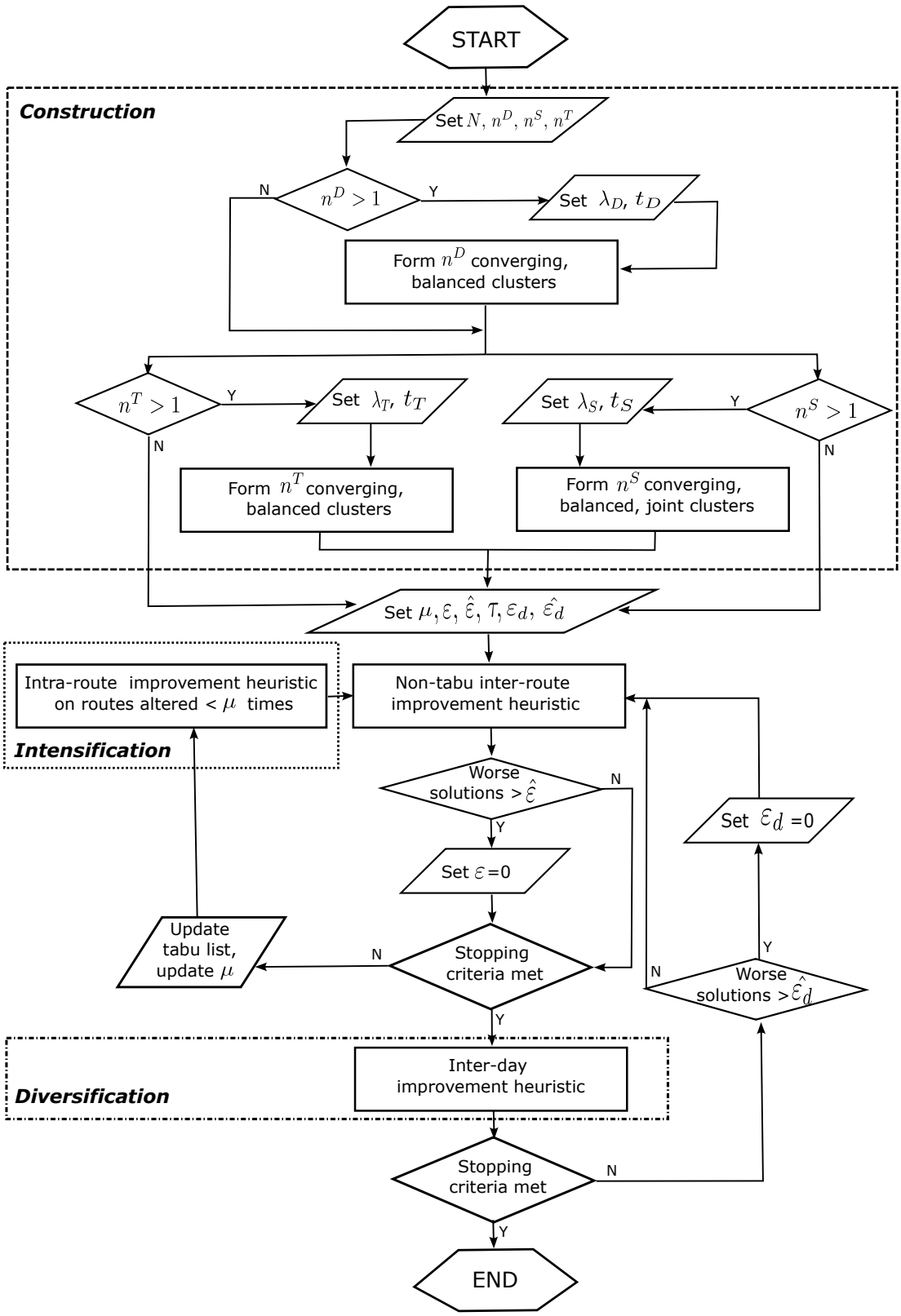


Figure 3.8: Tabu-search algorithm

Parameter fixing

After a series of computational experiments, for the study below the parameters described in Section 3.3 are fixed as follows:

$\lambda_D = \lambda_T = \lambda_S = 1$: Weights of centroids.

$t_D = t_T = t_S = 5$: Number of converging centroids.

$\mu = 1$: Route alteration limit for intra-route improvements.

$\varepsilon = 0.05$: Solution quality acceptance tolerance for inter-route improvements.

$\hat{\varepsilon} = 6$: Limit of worse solutions acceptable for inter-route improvements.

$\varepsilon_d = 5$: Solution quality acceptance tolerance for inter-day improvements.

$\hat{\varepsilon}_d = 0.01$: Limit of worse solutions acceptable for inter-day improvements.

$\tau = 10$: Tabu list length.

$t_{max} = 1h$: Execution time limit.

Construction phase

The construction phase is an important part of a heuristic, as a good quality initial solution requires lower improvement efforts. It is desirable to have a fast and well-performing construction heuristic because it can serve as a useful tool when a solution for a large instance is needed quickly.

To account for the randomness both the clustering and the routing stage, we ran all the algorithm five times for each instance, and recorded the best result observed. Table 3.1 – Table 3.6 show these results in the columns titled “Construction", while the best exact solution for each instance is described in the columns “Exact". We have recorded the solution, the optimality gap from the corresponding exact execution, and the run-time in seconds.

Instance properties					**Construction**		**Exact**		
Data set	n^C	n^T	n^S	n^D	**Sol**	**T(s)**	**Sol**	**Gap**	**T(s)**
20W1	20	2	2	3	9971	0.095	9202	32%	7200
20W1	20	2	2	2	8279	0.031	**7470**	0%	782
20W1	20	3	2	2	8718	0.031	8409	30%	7200
20W2	20	2	2	3	10172	0.073	9295	27%	7200
20W2	20	2	2	2	8496	0.043	7566	24%	7200
20W2	20	3	2	2	8954	0.048	8398	28%	7200
20W3	20	2	2	3	13084	0.067	12372	36%	7200
20W3	20	2	2	2	10652	0.035	9289	32%	7200
20W3	20	3	2	2	11552	0.088	10959	36%	7200
20E1	20	2	2	3	6717	0.063	6462	9%	7200
20E1	20	2	2	2	5715	0.038	5402	22%	7200
20E1	20	3	2	2	6303	0.058	6021	18%	7200
20E2	20	2	2	3	7454	0.043	7016	35%	7200
20E2	20	2	2	2	6224	0.039	5573	11%	7200
20E2	20	3	2	2	6645	0.051	6309	14%	7200
20E3	20	2	2	3	6287	0.055	6065	12%	7200
20E3	20	2	2	2	5475	0.037	5054	3%	7200
20E3	20	3	2	2	5844	0.044	5567	6%	7200
20P1	20	2	2	3	17025	0.079	14282	4%	7200
20P1	20	2	2	2	13849	0.039	**12990**	0%	3445
20P1	20	3	2	2	15250	0.039	14443	27%	7200

20P2	20	2	2	3	18806	0.043	15887	37%	7200
20P2	20	2	2	2	13942	0.047	13556	3%	7200
20P2	20	3	2	2	15782	0.047	15146	10%	7200
20P3	20	2	2	3	16769	0.054	14826	9%	7200
20P3	20	2	2	2	13204	0.031	**12613**	0%	5099
20P3	20	3	2	2	14554	0.035	14026	33%	7200
	Average				10582	0.050	9785	19%	6745

Table 3.1: Construction heuristic solutions for 20-customer instances

Instance properties					**Construction**		**Exact**		
Data set	n^C	n^T	n^S	n^D	**Sol**	**T(s)**	**Sol**	**Gap**	**T(s)**
25W1	25	2	2	3	11422	0.048	12086	50%	7200
25W1	25	2	2	2	9980	0.046	8854	39%	7200
25W1	25	3	2	2	11043	0.044	10081	44%	7200
25W2	25	2	2	3	8318	0.053	7636	8%	7200
25W2	25	2	2	2	7279	0.048	6561	3%	7200
25W2	25	3	2	2	7718	0.049	7208	7%	7200
25W3	25	2	2	3	9923	0.074	9748	29%	7200
25W3	25	2	2	2	7883	0.117	7703	42%	7200
25W3	25	3	2	2	8685	0.053	8181	45%	7200
25E1	25	2	2	3	6753	0.05	6692	29%	7200
25E1	25	2	2	2	5736	0.052	5293	18%	7200
25E1	25	3	2	2	6132	0.048	5786	22%	7200
25E2	25	2	2	3	5571	0.065	5944	31%	7200
25E2	25	2	2	2	4985	0.055	4744	8%	7200
25E2	25	3	2	2	5483	0.058	5138	9%	7200
25E3	25	2	2	3	6022	0.063	6184	21%	7200
25E3	25	2	2	2	4870	0.049	4856	9%	7200
25E3	25	3	2	2	5330	0.043	5021	6%	7200
25P1	25	2	2	3	16933	0.053	16878	46%	7200
25P1	25	2	2	2	13594	0.034	12378	2%	7200
25P1	25	3	2	2	14624	0.054	13808	4%	7200
25P2	25	2	2	3	18571	0.071	18250	46%	7200
25P2	25	2	2	2	14301	0.047	12996	2%	7200
25P2	25	3	2	2	15607	0.051	14648	6%	7200
25P3	25	2	2	3	17118	0.049	16799	51%	7200
25P3	25	2	2	2	13354	0.043	12230	2%	7200
25P3	25	3	2	2	14614	0.055	13650	3%	7200
	Average				10068	0.055	9606	21%	7200

Table 3.2: Construction heuristic solutions for 25-customer instances

Instance properties					**Construction**		**Exact**		
Data set	n^C	n^T	n^S	n^D	**Sol**	**T(s)**	**Sol**	**Gap**	**T(s)**
30W1	30	2	2	3	11681	0.069	12631	61%	7200
30W1	30	2	2	2	9140	0.044	8686	47%	7200
30W1	30	3	2	2	10252	0.044	11209	57%	7200
30W2	30	2	2	3	9425	0.056	9530	34%	7200
30W2	30	2	2	2	8136	0.037	6921	3%	7200
30W2	30	3	2	2	8378	0.059	7663	8%	7200
30W3	30	2	2	3	11259	0.051	11916	57%	7200

30W3	30	2	2	2	8645	0.105	8241	17%	7200
30W3	30	3	2	2	9826	0.046	10022	49%	7200
30E1	30	2	2	3	8349	0.063	8948	50%	7200
30E1	30	2	2	2	7122	0.048	6537	42%	7200
30E1	30	3	2	2	7483	0.048	7067	42%	7200
30E2	30	2	2	3	6544	0.062	6951	42%	7200
30E2	30	2	2	2	5350	0.052	4898	5%	7200
30E2	30	3	2	2	6085	0.072	5597	9%	7200
30E3	30	2	2	3	6645	0.074	6539	41%	7200
30E3	30	2	2	2	5410	0.051	4900	8%	7200
30E3	30	3	2	2	5946	0.063	5509	10%	7200
30P1	30	2	2	3	17374	0.059	17811	53%	7200
30P1	30	2	2	2	12837	0.065	**12002**	0%	39
30P1	30	3	2	2	14108	0.056	**13428**	0%	2313
30P2	30	2	2	3	24993	0.068	23639	54%	7200
30P2	30	2	2	2	19741	0.045	18954	49%	7200
30P2	30	3	2	2	20415	0.05	19770	20%	7200
30P3	30	2	2	3	18195	0.063	18577	50%	7200
30P3	30	2	2	2	13578	0.054	**12553**	0%	6030
30P3	30	3	2	2	15421	0.048	14232	4%	7200
Average					11198	0.057	10916	30%	6710

Table 3.3: Construction heuristic solutions for 30-customer instances

Instance properties					**Construction**		**Exact**		
Data set	n^C	n^T	n^S	n^D	**Sol**	**T(s)**	**Sol**	**Gap**	**T(s)**
35W1	35	2	2	3	12000	0.071	13561	34%	7200
35W1	35	2	2	2	10107	0.055	12413	46%	7200
35W1	35	3	2	2	10627	0.06	11859	42%	7200
35W2	35	2	2	3	9813	0.069	–	100%	7200
35W2	35	2	2	2	8509	0.056	7999	18%	7200
35W2	35	3	2	2	9293	0.065	8772	12%	7200
35W3	35	2	2	3	11708	0.073	–	100%	7200
35W3	35	2	2	2	10685	0.053	9978	13%	7200
35W3	35	3	2	2	11291	0.055	12181	26%	7200
35E1	35	2	2	3	8370	0.073	10185	40%	7200
35E1	35	2	2	2	6759	0.064	7088	40%	7200
35E1	35	3	2	2	7547	0.057	7481	39%	7200
35E2	35	2	2	3	7043	0.081	8552	35%	7200
35E2	35	2	2	2	6064	0.065	5682	11%	7200
35E2	35	3	2	2	6481	0.066	7515	33%	7200
35E3	35	2	2	3	8698	0.077	9384	55%	7200
35E3	35	2	2	2	6519	0.049	7240	46%	7200
35E3	35	3	2	2	7162	0.07	8835	53%	7200
35P1	35	2	2	3	22999	0.063	26920	69%	7200
35P1	35	2	2	2	17418	0.06	20425	38%	7200
35P1	35	3	2	2	19355	0.079	22106	34%	7200
35P2	35	2	2	3	20656	0.068	–	100%	7200
35P2	35	2	2	2	16112	0.06	16366	53%	7200
35P2	35	3	2	2	17628	0.06	17163	22%	7200
35P3	35	2	2	3	19042	0.06	22033	31%	7200
35P3	35	2	2	2	15056	0.045	13651	5%	7200
35P3	35	3	2	2	16303	0.06	15188	7%	7200
Average					11972	0.063	11207	41%	7200

Table 3.4: Construction heuristic solutions for 35-customer instances

Instance properties					Construction		Exact		
Data set	n^C	n^T	n^S	n^D	**Sol**	**T(s)**	**Sol**	**Gap**	**T(s)**
40W1	40	2	2	2	10675	0.064	11727	35%	7200
40W1	40	4	2	2	12788	0.09	13291	32%	7200
40W1	40	2	4	2	13756	0.071	13824	19%	7200
40W2	40	2	2	2	11004	0.06	12866	43%	7200
40W2	40	4	2	2	13371	0.089	15008	46%	7200
40W2	40	2	4	2	13760	0.058	15466	44%	7200
40W3	40	2	2	2	11254	0.067	10938	35%	7200
40W3	40	4	2	2	13305	0.067	16959	53%	7200
40W3	40	2	4	2	14269	0.057	14777	44%	7200
40E1	40	2	2	2	6515	0.063	5539	4%	7200
40E1	40	4	2	2	7261	0.107	8577	33%	7200
40E1	40	2	4	2	7788	0.08	8614	31%	7200
40E2	40	2	2	2	6243	0.067	6087	34%	7200
40E2	40	4	2	2	7689	0.077	8897	51%	7200
40E2	40	2	4	2	7928	0.09	8813	47%	7200
40E3	40	2	2	2	5879	0.047	5672	16%	7200
40E3	40	4	2	2	6973	0.091	7982	51%	7200
40E3	40	2	4	2	7044	0.061	6904	42%	7200
40P1	40	2	2	2	14823	0.062	14743	47%	7200
40P1	40	4	2	2	18331	0.063	19037	56%	7200
40P1	40	2	4	2	20417	0.054	23477	60%	7200
40P2	40	2	2	2	16037	0.061	15286	41%	7200
40P2	40	4	2	2	18517	0.068	20659	52%	7200
40P2	40	2	4	2	21432	0.054	22944	56%	7200
40P3	40	2	2	2	15236	0.053	15678	42%	7200
40P3	40	4	2	2	19465	0.099	17958	11%	7200
40P3	40	2	4	2	20813	0.054	22133	50%	7200
	Average				12688	0.069	13476	40%	7200

Table 3.5: Construction heuristic solutions for 40-customer instances

Instance properties					Construction		Exact		
Data set	n^C	n^T	n^S	n^D	**Sol**	**T(s)**	**Sol**	**Gap**	**T(s)**
45W1	45	2	2	2	8677	0.061	10573	49%	7200
45W1	45	4	2	2	11997	0.06	13915	58%	7200
45W1	45	2	4	2	12064	0.075	13783	56%	7200
45W2	45	2	2	2	12461	0.067	13572	49%	7200
45W2	45	4	2	2	14366	0.098	18938	49%	7200
45W2	45	2	4	2	15400	0.086	18829	42%	7200
45W3	45	2	2	2	10048	0.067	9185	27%	7200
45W3	45	4	2	2	11189	0.085	13222	43%	7200
45W3	45	2	4	2	11177	0.077	14415	33%	7200
45E1	45	2	2	2	6150	0.066	6573	45%	7200
45E1	45	4	2	2	6690	0.092	8088	39%	7200
45E1	45	2	4	2	7032	0.099	7450	48%	7200
45E2	45	2	2	2	6680	0.069	6661	19%	7200
45E2	45	4	2	2	7671	0.087	8892	33%	7200
45E2	45	2	4	2	7988	0.1	7722	16%	7200

45E3	45	2	2	2	6220	0.067	7376	41%	7200
45E3	45	4	2	2	7762	0.089	8859	35%	7200
45E3	45	2	4	2	7969	0.083	10508	54%	7200
45P1	45	2	2	2	16080	0.083	20132	53%	7200
45P1	45	4	2	2	19213	0.092	24776	58%	7200
45P1	45	2	4	2	20605	0.07	21783	25%	7200
45P2	45	2	2	2	17572	0.078	19008	53%	7200
45P2	45	4	2	2	22560	0.069	27653	45%	7200
45P2	45	2	4	2	21262	0.088	23656	33%	7200
45P3	45	2	2	2	17484	0.061	17655	50%	7200
45P3	45	4	2	2	21869	0.105	20833	28%	7200
45P3	45	2	4	2	21934	0.072	24117	62%	7200
Average					12967	0.079	14747	42%	7200

Table 3.6: Construction heuristic solutions for 45-customer instances

In the smaller instance groups, the exact method produces a better solution in most cases. However, in the slightly larger instance groups (30 – 45 customers), the heuristic results in a better solution and takes only a fraction of the time compared to the exact case. In Figure 3.9 the dashed line represents the trend of decreasing relative difference between the solutions obtained. This difference is calculated as $(Sol_H - Sol_E)/Sol_H$, and the time change is T_H/T_E, where the subscript H stands for Heuristic, and E for Exact. The negative values mean that the heuristic solution is better (lower) than the exact one. From the time graph, it can be seen that the heuristic on average takes less than 0.01% of the time required by the exact algorithm.

Compared to the best exact approach with a limit of 2 hours over all 162 instances, the construction heuristic alone found a better solution in 43% of the cases with an average solution improvement of 13% in under 0.01 seconds. In the cases where the construction heuristic results in a solution worse than that found with the exact method, the average gap between the two solutions is 2%.

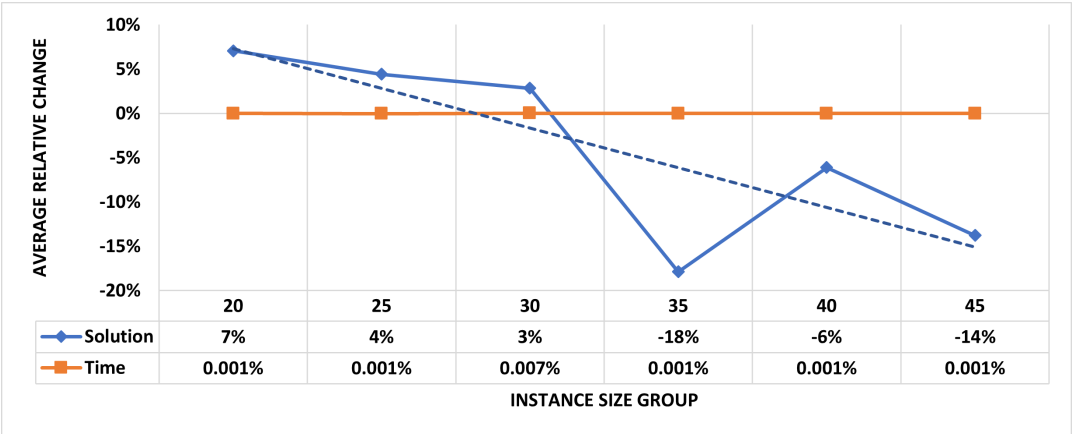


Figure 3.9: Solution and run-time for exact vs construction heuristic

Complete heuristic

As in the previous section, here we analyse the complete tabu-search algorithm. In Table 3.7 – Table 3.12, the properties of the heuristic algorithm are under the

column “Heuristic", and the best solution, optimality gap, and run-time from the exact analysis are under the column “Exact". The properties of the heuristic algorithm are given as follows:

- **Sol**: final solution.
- **T(s)**: overall run-time in seconds.
- ε: number of solutions worse than the limit $\hat{\varepsilon}$.
- $\mathbf{I_{div}}$: number of successful diversification iterations.
- $\mathbf{T_{div}}$: time taken by the diversification procedure in seconds.
- **I**: number of tabu-search iterations.
- **Gap**: Solution gap between the upper and lower bounds from the best exact solution.

Instance properties					Heuristic						Exact		
Data set	n^C	n^T	n^S	n^D	**Sol**	**T(s)**	ε	$\mathbf{I_{div}}$	$\mathbf{T_{div}}$	**I**	**Sol**	**Gap**	**T(s)**
20W1	20	2	2	3	9993	12.509	5	0	6.376	7	9202	32%	7200
20W1	20	2	2	2	8191	10.326	4	0	2.988	8	7470	0%	782
20W1	20	2	3	2	8721	17.782	6	0	3.383	17	8409	30%	7200
20W2	20	2	2	3	9782	11.257	6	0	3.716	17	9295	27%	7200
20W2	20	2	2	2	8334	10.023	6	0	2.123	18	7566	24%	7200
20W2	20	2	3	2	8939	12.61	6	0	2.702	12	8398	28%	7200
20W3	20	2	2	3	13084	6.499	6	0	3.785	9	12372	36%	7200
20W3	20	2	2	2	10245	10.653	6	0	1.984	18	9289	32%	7200
20W3	20	2	3	2	10947	14.275	6	0	2.183	19	10959	36%	7200
20E1	20	2	2	3	6576	5.93	6	0	2.922	13	6462	9%	7200
20E1	20	2	2	2	5616	7.43	6	0	2.036	11	5402	22%	7200
20E1	20	2	3	2	6228	13.73	6	0	3.06	16	6021	18%	7200
20E2	20	2	2	3	7513	5.648	6	0	3.341	11	7016	35%	7200
20E2	20	2	2	2	5734	11.146	6	0	2.39	17	5573	11%	7200
20E2	20	2	3	2	6459	11.568	6	0	2.213	19	6309	14%	7200
20E3	20	2	2	3	6188	18.283	11	2	10.911	22	6065	12%	7200
20E3	20	2	2	2	5419	7.205	6	0	1.919	12	5054	3%	7200
20E3	20	2	3	2	5662	14.91	6	0	2.596	13	5567	6%	7200
20P1	20	2	2	3	16771	22.569	16	3	10.004	37	14282	4%	7200
20P1	20	2	2	2	13548	7.777	6	0	2.006	12	12990	0%	3445
20P1	20	2	3	2	15090	11.669	6	0	2.197	15	14443	27%	7200
20P2	20	2	2	3	18439	6.48	6	0	4.015	13	15887	37%	7200
20P2	20	2	2	2	13814	8.19	6	0	2.004	13	13556	3%	7200
20P2	20	2	3	2	15266	8.253	6	0	2.292	12	15146	10%	7200
20P3	20	2	2	3	16526	6.673	6	0	3.351	13	14826	9%	7200
20P3	20	2	2	2	12810	8.405	6	0	2.167	14	12613	0%	5099
20P3	20	2	3	2	14231	8.132	6	0	2.616	11	14026	33%	7200

Table 3.7: Heuristic solutions for 20-customer instances

Instance properties					Heuristic						Exact		
Data set	n^C	n^T	n^S	n^D	**Sol**	**T(s)**	ε	$\mathbf{I_{div}}$	$\mathbf{T_{div}}$	**I**	**Sol**	**Gap**	**T(s)**
25W1	25	2	2	3	11088	9.464	6	0	5.245	10	12086	50%	7200
25W1	25	2	2	2	9444	18.127	6	0	3.444	19	8854	39%	7200
25W1	25	2	3	2	10565	21.35	6	0	4.074	17	10081	44%	7200

25W2	25	2	2	3	8130	11.338	6	0	5.466	14	7636	8%	7200
25W2	25	2	2	2	6949	16.324	6	0	3.603	17	6561	3%	7200
25W2	25	2	3	2	7303	15.862	6	0	4.346	13	7208	7%	7200
25W3	25	2	2	3	9481	10.249	6	0	5.154	15	9748	29%	7200
25W3	25	2	2	2	7184	11.75	3	0	4.097	8	7703	42%	7200
25W3	25	2	3	2	8336	16.311	6	0	4.224	13	8181	45%	7200
25E1	25	2	2	3	6656	9.858	6	0	5.416	10	6692	29%	7200
25E1	25	2	2	2	5532	15.202	6	0	3.567	16	5293	18%	7200
25E1	25	2	3	2	5882	15.943	6	0	4.146	14	5786	22%	7200
25E2	25	2	2	3	5539	6.374	0	0	5.318	1	5944	31%	7200
25E2	25	2	2	2	4793	12.346	4	0	3.973	9	4744	8%	7200
25E2	25	2	3	2	5448	15.769	6	0	4.308	13	5138	9%	7200
25E3	25	2	2	3	5907	12.644	6	0	5.569	15	6184	21%	7200
25E3	25	2	2	2	4771	8.251	3	0	3.574	5	4856	9%	7200
25E3	25	2	3	2	5153	13.358	6	0	4.638	11	5021	6%	7200
25P1	25	2	2	3	16570	13.468	6	0	5.757	17	16878	46%	7200
25P1	25	2	2	2	12686	17.555	6	0	3.468	14	12378	2%	7200
25P1	25	2	3	2	13932	18.425	6	0	4.396	15	13808	4%	7200
25P2	25	2	2	3	18023	13.895	6	0	7.242	15	18250	46%	7200
25P2	25	2	2	2	13289	16.471	6	0	3.398	17	12996	2%	7200
25P2	25	2	3	2	14705	20.557	6	0	4.051	17	14648	6%	7200
25P3	25	2	2	3	16811	12.985	6	0	5.679	15	16799	51%	7200
25P3	25	2	2	2	12673	21.381	6	0	3.706	23	12230	2%	7200
25P3	25	2	3	2	14040	19.701	6	0	4.29	17	13650	3%	7200

Table 3.8: Heuristic solutions for 25-customer instances

Instance properties					**Heuristic**						**Exact**		
Data set	n^C	n^T	n^S	n^D	**Sol**	**T(s)**	ε	$\mathbf{I_{div}}$	$\mathbf{T_{div}}$	**I**	**Sol**	**Gap**	**T(s)**
30W1	30	2	2	3	11309	17.939	6	0	8.821	13	12631	61%	7200
30W1	30	2	2	2	8724	19.103	6	0	6.042	10	8686	47%	7200
30W1	30	2	3	2	9785	37.95	6	0	7.142	23	11209	57%	7200
30W2	30	2	2	3	8999	18.93	6	0	8.763	15	9530	34%	7200
30W2	30	2	2	2	7080	26.036	6	0	6.164	17	6921	3%	7200
30W2	30	2	3	2	7573	28.359	6	0	6.927	16	7663	8%	7200
30W3	30	2	2	3	10860	21.309	6	0	8.345	21	11916	57%	7200
30W3	30	2	2	2	8033	26.808	6	0	5.935	17	8241	17%	7200
30W3	30	2	3	2	9183	27.819	6	0	6.805	15	10022	49%	7200
30E1	30	2	2	3	8211	21.975	6	0	9.487	13	8948	50%	7200
30E1	30	2	2	2	6498	30.859	6	0	7.231	18	6537	42%	7200
30E1	30	2	3	2	7163	28.696	6	0	6.796	17	7067	42%	7200
30E2	30	2	2	3	6340	16.784	6	0	8.245	13	6951	42%	7200
30E2	30	2	2	2	4910	26.217	6	0	5.937	17	4898	5%	7200
30E2	30	2	3	2	5609	31.527	6	0	7.134	18	5597	9%	7200
30E3	30	2	2	3	6444	29.396	6	0	14.351	16	6539	41%	7200
30E3	30	2	2	2	5180	27.598	6	0	6.809	13	4900	8%	7200
30E3	30	2	3	2	5598	34.127	6	0	8.17	17	5509	10%	7200
30P1	30	2	2	3	16928	21.038	6	0	8.472	19	17811	53%	7200
30P1	30	2	2	2	12526	21.981	6	0	5.952	13	12002	0%	39
30P1	30	2	3	2	13867	33.246	6	0	8.106	15	13428	0%	2313
30P2	30	2	2	3	23546	27.753	6	0	13.434	19	23639	54%	7200
30P2	30	2	2	2	18164	24.493	6	0	5.691	15	18954	49%	7200
30P2	30	2	3	2	19442	24.311	6	0	7.015	12	19770	20%	7200
30P3	30	2	2	3	17845	20.71	6	0	8.734	18	18577	50%	7200
30P3	30	2	2	2	12986	21.738	6	0	6.292	12	12553	0%	6030

30P3	30	2	3	2	14531	28.525	6	0	7.101	15	14232	4%	7200

Table 3.9: Heuristic solutions for 30-customer instances

Instance properties					Heuristic						Exact		
Data set	n^C	n^T	n^S	n^D	**Sol**	**T(s)**	ε	$\mathbf{I_{div}}$	$\mathbf{T_{div}}$	**I**	**Sol**	**Gap**	**T(s)**
35W1	35	2	2	3	11212	28.605	6	0	13.089	15	13561	34%	7200
35W1	35	2	2	2	9493	35.752	6	0	10.098	14	12413	46%	7200
35W1	35	2	3	2	10170	42.429	6	0	12.545	13	11859	42%	7200
35W2	35	2	2	3	9568	30.387	6	0	13.386	16	–	100%	7200
35W2	35	2	2	2	7680	33.962	6	0	9.207	14	7999	18%	7200
35W2	35	2	3	2	8562	56.23	6	0	11.135	23	8772	12%	7200
35W3	35	2	2	3	11232	39.528	6	0	14.97	21	–	100%	7200
35W3	35	2	2	2	10075	33.303	6	0	9.731	13	9978	13%	7200
35W3	35	2	3	2	10453	43.534	6	0	11.387	16	12181	26%	7200
35E1	35	2	2	3	8149	28.672	6	0	12.695	16	10185	40%	7200
35E1	35	2	2	2	6478	36.9	6	0	9.414	16	7088	40%	7200
35E1	35	2	3	2	7204	51.866	6	0	10.663	22	7481	39%	7200
35E2	35	2	2	3	6709	28.31	6	0	13.035	15	8552	35%	7200
35E2	35	2	2	2	5580	36.819	6	0	9.241	15	5682	11%	7200
35E2	35	2	3	2	5982	39.219	6	0	11.822	14	7515	33%	7200
35E3	35	2	2	3	8047	29.646	6	0	12.769	17	9384	55%	7200
35E3	35	2	2	2	6314	48.376	6	0	9.833	20	7240	46%	7200
35E3	35	2	3	2	6869	49.891	6	0	11.438	20	8835	53%	7200
35P1	35	2	2	3	22540	25.238	6	0	12.66	10	26920	69%	7200
35P1	35	2	2	2	16171	43.927	6	0	8.985	20	20425	38%	7200
35P1	35	2	3	2	18010	49.631	6	0	9.629	20	22106	34%	7200
35P2	35	2	2	3	20399	26.564	6	0	12.537	13	–	100%	7200
35P2	35	2	2	2	15584	74.473	12	1	15.582	30	16366	53%	7200
35P2	35	2	3	2	16467	40.655	6	0	10.501	15	17163	22%	7200
35P3	35	2	2	3	18649	32.928	6	0	14.756	13	22033	31%	7200
35P3	35	2	2	2	14130	37.885	6	0	9.172	16	13651	5%	7200
35P3	35	2	3	2	15474	37.045	6	0	10.705	13	15188	7%	7200

Table 3.10: Heuristic solutions for 35-customer instances

Instance properties					Heuristic						Exact		
Data set	n^C	n^T	n^S	n^D	**Sol**	**T(s)**	ε	$\mathbf{I_{div}}$	$\mathbf{T_{div}}$	**I**	**Sol**	**Gap**	**T(s)**
40W1	40	2	2	2	9547	64.65	6	0	15.273	21	11727	35%	7200
40W1	40	2	4	2	11454	80.575	6	0	20.756	20	13291	32%	7200
40W1	40	4	2	2	12880	68.246	6	0	18.909	17	13824	19%	7200
40W2	40	2	2	2	10479	42.549	6	0	13.226	14	12866	43%	7200
40W2	40	2	4	2	12336	63.992	6	0	23.243	14	15008	46%	7200
40W2	40	4	2	2	13345	50.456	6	0	17.05	13	15466	44%	7200
40W3	40	2	2	2	10591	50.661	6	0	14.529	14	10938	35%	7200
40W3	40	2	4	2	12695	50.47	6	0	18.239	12	16959	53%	7200
40W3	40	4	2	2	13741	57.06	6	0	17.104	15	14777	44%	7200
40E1	40	2	2	2	6310	61.422	6	0	19.265	13	5539	4%	7200
40E1	40	2	4	2	6913	72.788	6	0	23.069	18	8577	33%	7200
40E1	40	4	2	2	7397	62.853	6	0	18.624	15	8614	31%	7200
40E2	40	2	2	2	5400	65.625	6	0	13.439	23	6087	34%	7200
40E2	40	2	4	2	7200	58.826	6	0	19.692	14	8897	51%	7200

40E2	40	4	2	2	7618	57.009	6	0	18.383	13	8813	47%	7200
40E3	40	2	2	2	5288	49.541	6	0	12.115	18	5672	16%	7200
40E3	40	2	4	2	6133	70.384	6	0	18.224	19	7982	51%	7200
40E3	40	4	2	2	6729	51.713	6	0	15.593	14	6904	42%	7200
40P1	40	2	2	2	13160	51.318	6	0	15.165	13	14743	47%	7200
40P1	40	2	4	2	16894	70.968	6	0	16.905	21	19037	56%	7200
40P1	40	4	2	2	19325	71.944	6	0	17.189	18	23477	60%	7200
40P2	40	2	2	2	14477	72.45	6	0	14.576	24	15286	41%	7200
40P2	40	2	4	2	17626	64.536	6	0	19.954	15	20659	52%	7200
40P2	40	4	2	2	20211	71.461	6	0	17.929	17	22944	56%	7200
40P3	40	2	2	2	14588	63.302	6	0	14.104	19	15678	42%	7200
40P3	40	2	4	2	17940	69.512	6	0	17.93	18	17958	11%	7200
40P3	40	4	2	2	19817	77.542	6	0	17.463	20	22133	50%	7200

Table 3.11: Heuristic solutions for 40-customer instances

Instance properties					**Heuristic**						**Exact**		
Data set	n^C	n^T	n^S	n^D	**Sol**	**T(s)**	ε	$\mathbf{I_{div}}$	$\mathbf{T_{div}}$	**I**	**Sol**	**Gap**	**T(s)**
45W1	45	2	2	2	8665	55.972	6	0	20.164	10	10573	49%	7200
45W1	45	2	4	2	11201	78.175	6	0	26.855	15	13915	58%	7200
45W1	45	4	2	2	10990	62.908	6	0	23.708	11	13783	56%	7200
45W2	45	2	2	2	11065	68.211	6	0	20.218	15	13572	49%	7200
45W2	45	2	4	2	13193	92.451	6	0	27.333	18	18938	49%	7200
45W2	45	4	2	2	14592	72.944	6	0	26.342	12	18829	42%	7200
45W3	45	2	2	2	8700	61.152	6	0	19.571	15	9185	27%	7200
45W3	45	2	4	2	10263	92.604	6	0	30.586	16	13222	43%	7200
45W3	45	4	2	2	10602	66.367	6	0	26.931	10	14415	33%	7200
45E1	45	2	2	2	5408	91.638	6	0	20.798	23	6573	45%	7200
45E1	45	2	4	2	6263	90.719	6	0	28.287	17	8088	39%	7200
45E1	45	4	2	2	6606	79.315	6	0	26.656	14	7450	48%	7200
45E2	45	2	2	2	6072	64.724	6	0	19.331	15	6661	19%	7200
45E2	45	2	4	2	7105	85.244	6	0	25.6	17	8892	33%	7200
45E2	45	4	2	2	7578	83.662	6	0	24.078	16	7722	16%	7200
45E3	45	2	2	2	6000	58.787	6	0	20.343	12	7376	41%	7200
45E3	45	2	4	2	7367	112.803	6	0	31.678	16	8859	35%	7200
45E3	45	4	2	2	7790	66.195	6	0	26.252	10	10508	54%	7200
45P1	45	2	2	2	15416	58.3	6	0	19.39	12	20132	53%	7200
45P1	45	2	4	2	17573	67.964	6	0	28.57	10	24776	58%	7200
45P1	45	4	2	2	19071	88.459	6	0	25.685	16	21783	25%	7200
45P2	45	2	2	2	16342	65.875	6	0	19.935	14	19008	53%	7200
45P2	45	2	4	2	20014	198.816	12	1	51.04	40	27653	45%	7200
45P2	45	4	2	2	20524	94.35	6	0	26.016	18	23656	33%	7200
45P3	45	2	2	2	15456	75.098	6	0	23.81	12	17655	50%	7200
45P3	45	2	4	2	19839	80.9	6	0	27.005	14	20833	28%	7200
45P3	45	4	2	2	20069	139.542	12	1	48.748	22	24117	62%	7200

Table 3.12: Heuristic solutions for 45-customer instances

In Table 3.13 we have summarised the results by instance group size.

The column “RC sols" gives the average relative change between the heuristic and the best solution obtained by an exact algorithm, calculated as $(Sol_H - Sol_E)/Sol_H$. In the smaller instances, the exact solution is better on average, while in the larger instances (greater than 30 customers) the heuristic solution is better. Overall, the heuristic solution is better by 6%.

The time gaps between the run-time of the heuristic and the exact solution are in the column “RC time". The heuristic algorithm on average requires 1% of the exact execution time with an imposed limit of 2 hours.

From the column “T_{Div}", which stands for the time consumed by the diversification procedure, it can be seen that a rather large portion of time is dedicated to this step of the algorithm. However, column “Num div" which gives the overall number of times the diversification procedure resulted in an improvement, shows that this procedure is often too costly. In the instance groups of 25 and 40 customers, no improvement is found with the diversification procedure, even though this procedure explores all possible inter-day swaps. This indicates the good performance of the clustering procedures at the construction stage.

In column “$\hat{\varepsilon}$ reached" we show the number of worse solutions accepted during the tabu-search process. The values represent the frequency with which the number of worse solutions accepted is the maximum number of worse solutions allowed ($\hat{\varepsilon}$). In these experiments, this value was set to 6, and in 2 of the size groups this value was not reached (on average).

Column “Avrg iter" shows the average number of tabu-search iterations. The stopping criteria is only the point when no improvement can be found, even with the worse solution allowance of $\hat{\varepsilon} = 6$. The overall average is 15 iterations.

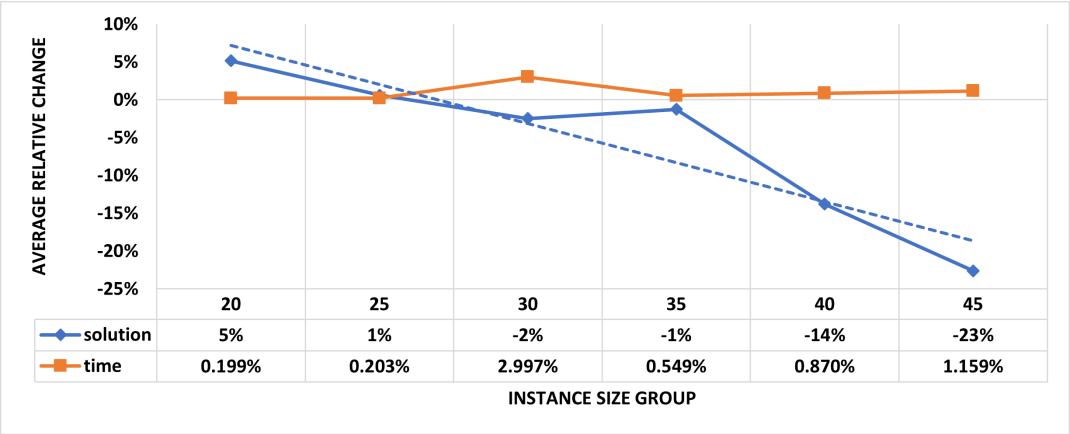
The final column titled “Tabu improvement" shows the overall solution improvement from the solution obtained by the construction heuristic, using the implementation of the 3 improvement methods from Section 3.2. The improvement is calculated as the relative difference between the construction heuristic solution and the tabu-search solution $((Sol_{const} - Sol_{tabu})/Sol_{const})$. This improvement increases monotonously with the increase of the instance size, which promises that this procedure is valuable for larger instances to a certain extent.

Instance size	RC sols	RC time	T_{Div}	Num div	$\hat{\varepsilon}$ reached	Avrg iter	Tabu improvement
20	5%	0.1%	33%	7%	85%	14.8	2%
25	1%	0.2%	35%	0%	89%	13.7	4%
30	-2%	3.1%	38%	4%	100%	15.6	6%
35	-1%	0.5%	31%	4%	100%	16.7	6%
40	-14%	0.9%	28%	0%	100%	16.7	7%
45	-23%	1.2%	32%	7%	100%	15.6	8%
Average	-6%	1%	33%	4%	96%	15.50	5%

Table 3.13: Properties of heuristic results

The relative change between the solution obtained by the heuristic and by an exact method, as well as the time gap is shown in Figure 3.10. The trend shown with a dashed line shows a clear increase in solution change in favour of the heuristic, while the time change line remains close to 0%.

Similarly as in Section 2.9.2, using Dolan and Moré's performance profiles [39] and the upper bound as a performance metric, we obtain a result that clearly favours the heuristic method as shown in Figure 3.11.



	20	25	30	35	40	45
solution	5%	1%	-2%	-1%	-14%	-23%
time	0.199%	0.203%	2.997%	0.549%	0.870%	1.159%

Figure 3.10: Solution and run-time for exact vs heuristic algorithm

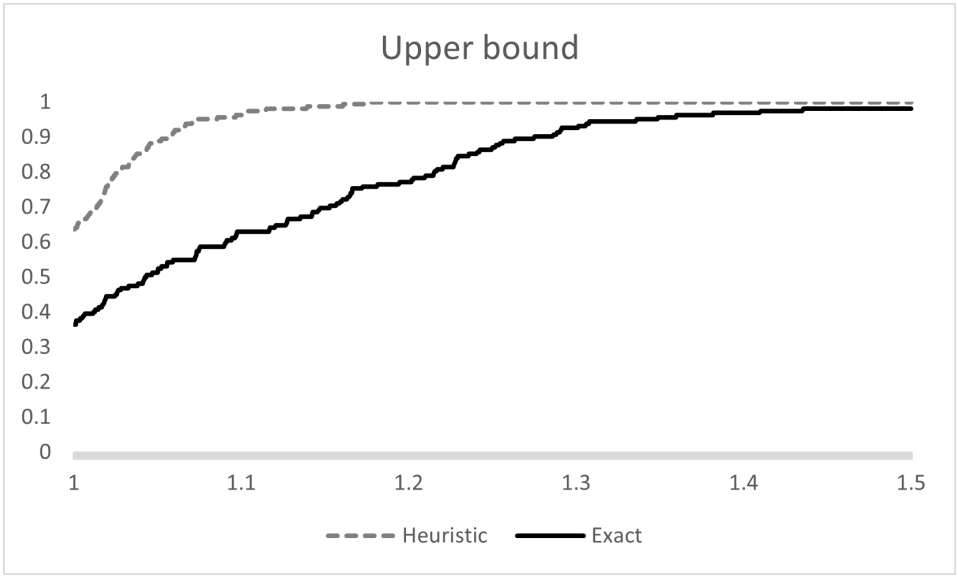


Figure 3.11: Performance profiles of the exact method and heuristic

3.5 Original problem instance

Considering the performance analysis of the solution methods for the two-stage multi-period VRP with depot location and route balancing, we will now provide a solution using the original data set provided by the company that inspired the problem. The data set consists of 1,756 customers and a fixed depot.

From the results presented in Section 2.9.2 it is safe to say that an instance of this size would be infeasible to try to solve with an exact algorithm. Both the 3- and 4-index models will result in an extreme number of binary variables and complicating constraints.

We adjust the heuristic algorithm to solve this instance. From the results shown in Section 3.4 we consider the following assumptions:

1. **Tabu-search.** The evaluation of all possible customer swaps within the graph is costly. For this reason, we perform a random permutation of the customer set during each iteration, evaluating only the first η_t customer swaps. Once

a non-tabu improving move is identified, the change is implemented in the corresponding routes, the tabu list is updated and the algorithm moves to the next iteration. The procedure continues until no improvement can be identified in this step.

2. **Intensification.** Investigating all possible pairs of edges within every route for both sellers and trucks is also very costly. Instead, we permute the order of the edges and form random pairs in every execution, while only evaluating the first η_i swaps. As soon as a beneficial swap is found, the route is adjusted and the procedure ends.

3. **Diversification.** From the previous analysis we know that the diversification procedure will take a large portion of run-time, without a meaningful benefit for a problem of this magnitude. For this reason, the diversification step will be omitted.

4. **Stopping criterion.** The stopping criterion for the complete algorithm is the point when no improvement is identified during an iteration. No maximum number of iterations is imposed.

	T_{const}	T_{tabu}	Sol_{const}	Sol_{tabu}
Run 1	10.05	677.78	99911	98381
Run 2	9.67	792.91	99765	98356
Run 3	9.055	769.84	99744	98453
Run 4	**11.731**	**822.67**	**97599**	**96785**
Run 5	11.326	500.44	98631	97899
Average	10.3664	712.728	99130	97974.8

Table 3.14: 5 runs of the original problem

The algorithm was run 5 times with $\eta_t = \eta_i = 50$, while the construction phase remains as defined for the test instances. The results are given in Table 3.14 and the best execution is given in bold. The column titled "T_{const}" shows the construction time in seconds, and the column "T_{tabu}" shows the total execution time of the tabu-search algorithm. The remaining 2 columns show the solution from the construction phase (Sol_{const}) and the final solution (Sol_{tabu}). In Figure 3.12 we show the solution obtained during the first 10 iterations of each run. The average number of iterations across all 5 runs is 13. The average execution time of this algorithm is 11 minutes, while the construction heuristic alone averages at 10 seconds. This is a significant difference in time, granting an improvement of at least 1%. This result can be improved with the cost of longer run-times and higher computational power if the parameters η_t and η_i are higher and the intensification procedures are enhanced to continue the search after the first improvement.

Figure 3.13 shows the final solution of the original problem with 1,756 customers, 5 days, 2 trucks and 10 sellers. Figures (a) and (b) show the complete routing colour coded by day clusters. Figures (c) and (d) single out a single day cluster, and colour code the truck and seller routes within, respectively.

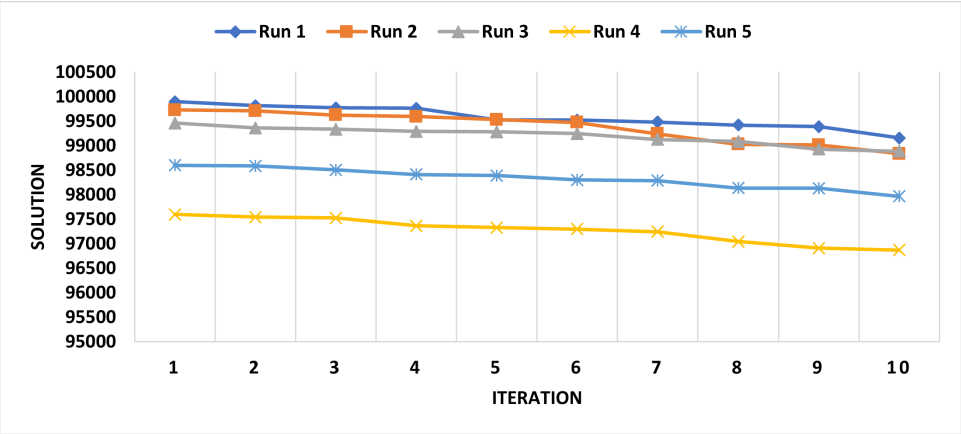
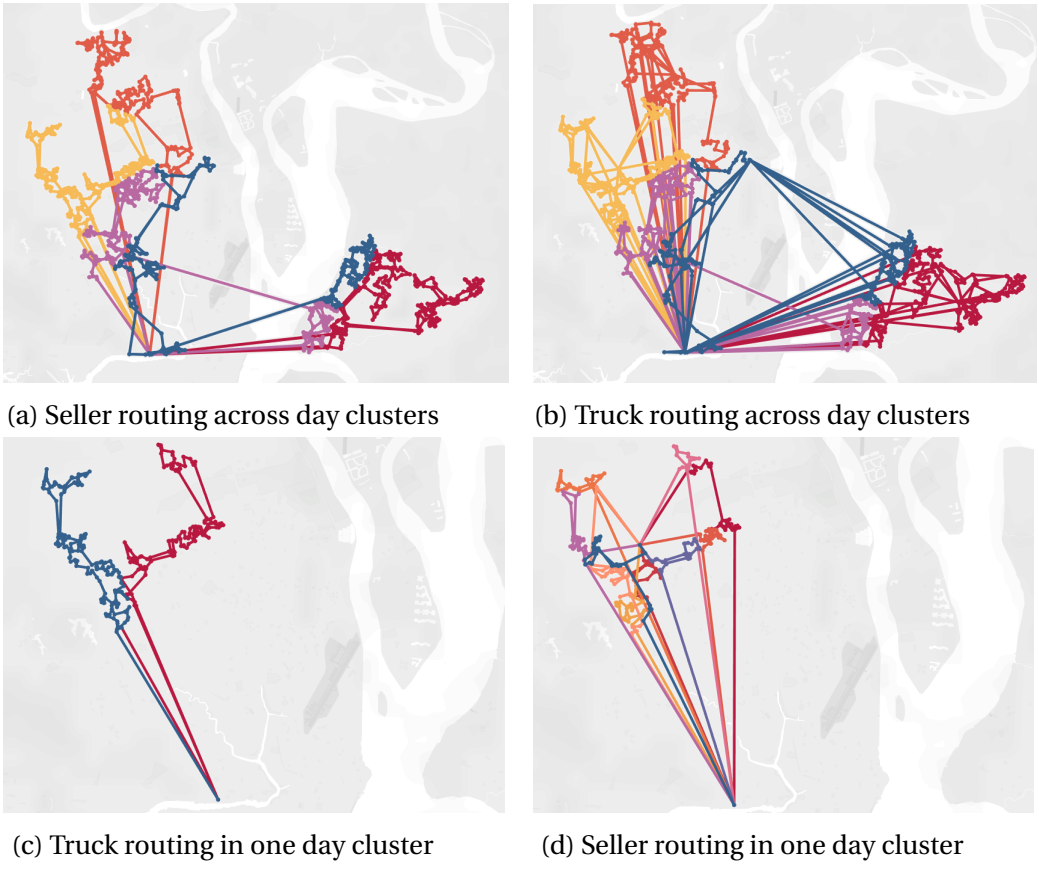


Figure 3.12: Complete solution after 5 runs



(a) Seller routing across day clusters

(b) Truck routing across day clusters

(c) Truck routing in one day cluster

(d) Seller routing in one day cluster

Figure 3.13: Original problem solution illustration

3.6 Conclusions

Due to the size of the original problem, attempting either of the proposed exact approaches to obtain at least a feasible solution is intractable. For this reason, we propose a fast and powerful heuristic procedure based on a tabu-search framework. We propose a cluster-first route-second construction heuristic that makes use of a machine learning clustering algorithm. We propose 3 alterations to this algorithm: i)

multi-centroid clustering resulting in clusters that converge towards a given point, ii) *balanced clustering* resulting in balanced clusters, iii) *joint clustering* resulting in clusters that intersect. These procedures can be used individually or combined to fit a wide variety of VRPs. For the routing stage, a random best insertion is performed. To the best of our knowledge, a similar procedure cannot be found in the literature. This construction procedure is shown to be very fast and powerful, resulting in a better solution than the best-known in 43% of the instances in less than 0.1 seconds, with an average improving solution gap of 13%.

We implement 3 improvement heuristics within a tabu-search framework, through which the initial solution obtained with the construction heuristic is on average improved by 5%. The final results show an average improvement of 6% over the best-known solution obtained with an exact method. The computational experiments performed for the analysis of the proposed heuristic were performed 5 times to account for the randomness of the procedures, and they involved a total of 9 hours of computational time approximately.

Finally, the original problem is solved using our proposed heuristic. The construction stage provided a feasible solution in 10 seconds, and the tabu search stage improved upon it by 1%. The run-time of the complete heuristic was approximately 12 minutes.

Chapter 4

A solution approach for multi-trip vehicle routing problems with time windows, fleet sizing and depot location

In this chapter we present a case study proposed for the 12th AIMMS-MOPTA Optimization Modeling Competition [2]. It is a variant of the Vehicle Routing Problem with some additional elements involving the number and location of the depots, number and types of routes allowed for the vehicles and specific customer needs. Extensive research was conducted during the course of four months. This is joint work by the members of the competition's winning team: Ivona Gjeroska[1], Paula Fermín Cueto[1], and Albert Solà Vilalta[1]. The project was supervised by Professor Miguel F. Anjos[1]. The work presented in this chapter is published in [30].

4.1 Introduction

In this case study we present a variant of the CVRP with some additional elements.

- It is required that every customer be visited within a given time window (8am and 4pm) and, additionally, vehicles can only operate within fixed time intervals (from 6am to 5pm).
- In this problem, there is not a single depot; a fixed number of depots is given as a problem parameter; the locations of these depots must be determined.
- The size of the fleet assigned to each depot must be minimised and this assignment must remain constant for the entire duration of the problem.
- Vehicles are allowed to perform multiple trips, starting and terminating at the same depot. We use the term *trip* in the context of the multi-trip aspect of the problem: it refers to an occasion where a given vehicle leaves the depot to visit

[1] University of Edinburgh, School of Mathematics

customers and return. By contrast, we use the term *route* to refer to the actual path that a vehicle follows during a trip.

Even though all customers have the same time window, the definition of time windows cannot be avoided due to the difference between the opening and closing times of the depot. The vehicles have two additional hours to reach each customer in the morning before their opening times, and one additional hour to reach the depot after their closing times. Each of the problems listed above fall under a certain VRP variant (VRPTW, MDVRP and LRP, FSP, and MTVRP, respectively) defined in Section 1.3. All of the variants mentioned are known to be computationally challenging, and heuristic methods have been developed as a consequence of the limitations of the exact approaches.

The data provided for this problem as part of the 12th AIMMS-MOPTA Optimization Modeling Competition [2] includes a graph of 1,457 nodes representing ZIP codes in the state of Pennsylvania, USA. The given graph is not complete, but it is connected, so a path can be found between any two nodes using existing edges.

Demand of each customer is provided for every day in 2018 and 2019. Despite the large number of customers, on average only 5% must be visited on a given day. The 2018 data is used for model development while the 2019 data is reserved for testing the robustness of our solution method. In practice, demand for a given day is not known until 6pm the day before, so all routes should be obtained quickly enough to inform the drivers on time.

We propose a method that efficiently determines the location of a given number of depots, optimizes the fleet size and produces a sensible routing strategy in a reasonable time.

This chapter is structured as follows: in Section 4.2 we present a mathematical formulation of the problem. Our methodology is outlined in Section 4.3. Sections 4.4 and 4.5 present an analysis of the minimum number of depots and the depot location model, respectively. In Section 4.6, a robust approach to the fleet sizing problem is proposed. Section 4.7 describes an algorithm to assign customers to depots on a daily basis. Both exact and heuristic approaches for the routing problem are described in Section 4.8. Results are presented in Section 4.9. We present a methodology for automating the choice of the number of depots in Section 4.10 and lastly, a conclusion is given in Section 4.11.

4.2 Mathematical formulation

In this section we present a mixed integer linear programming formulation for the multi-trip VRP with time windows, fleet sizing and depot location. Some assumptions were made at different stages of the problem:

- Each customer must be served by exactly one vehicle.
- The demand of customers co-located with a depot is assumed to be served automatically.

- The distance between two customers is the shortest path between their locations through known routes. The Dijkstra algorithm [38] was applied to generate a distance matrix with all the shortest paths between any two nodes in the network.
- No time is needed to unload the demand at a customer location or to reload at the depot.

We define an undirected graph $G = (N, E)$ where the set of nodes N represents the customers and the set of edges E represents the shortest paths between customers. We also define a set of unknown depot locations, a set of potential vehicles to be used and a set of trips each vehicle might take. Formally, we introduce the following mathematical formulation:

Sets:

N : set of nodes.
E : set of edges, i.e., $E = N \times N$.
$H \subseteq N$: set of depots.
K : set of vehicles.
R : set of trips per vehicle.

Parameters:

f_i : cost of setting a depot on location $i \in N$.
c_{ij} : cost of traversing edge $(i, j) \in E$.
u : cost of a vehicle.
Q : capacity of a vehicle.
q_i : demand at node $i \in N$.
t_{ij} : time required to travel from $i \in N$ to $j \in N$.
$[a^c, b^c]$: time window of customers.
$[a^d, b^d]$: depot opening hours.

Variables:

$$x_{ij}^{kr} = \begin{cases} 1, & \text{if vehicle } k \text{ traverses edge } (i, j) \text{ in trip } r \\ 0, & \text{otherwise.} \end{cases}$$

$$z_i = \begin{cases} 1, & \text{if there is a depot at node } i \\ 0, & \text{otherwise.} \end{cases}$$

$$w_i^k = \begin{cases} 1, & \text{if vehicle } k \text{ is assigned to a depot at node } i \\ 0, & \text{otherwise.} \end{cases}$$

s_i : time node i is visited (if not a depot).

h_1^{kr} : time the depot is visited by its assigned vehicle $k$at the start of trip r.

h_2^{kr} : time the depot is visited by its assigned vehicle $k$at the end of trip r.

$$\min \sum_{i\in V}\sum_{j\in V} c_{ij} \sum_{k\in K}\sum_{r\in R} x_{ij}^{kr} + \sum_{i\in V} f_i z_i + u \sum_{k\in K}\sum_{i\in V} w_i^k$$

s.t.

$$\sum_{j\in V}\sum_{k\in K}\sum_{r\in R} x_{ji}^{kr} \geq 1 \qquad \forall\, i \in N, \tag{4.1}$$

$$\sum_{j\in N}\sum_{k\in K}\sum_{r\in R} x_{ji}^{kr} \leq 1 + |N|\, z_i \qquad \forall\, i \in N, \tag{4.2}$$

$$\sum_{j\in N} x_{ij}^{kr} = \sum_{j\in N} x_{ji}^{kr} \qquad \forall\, i \in V, k \in K, r \in R, \tag{4.3}$$

$$\sum_{j\in V} x_{ij}^{kr} \leq 1 \qquad \forall\, i \in V, k \in K, r \in R, \tag{4.4}$$

$$\sum_{i\in V} q_i (\sum_{j\in V} x_{ij}^{kr} - w_i^k) \leq Q \qquad \forall\, k \in K, r \in R, \tag{4.5}$$

$$\sum_{i\in V} z_i = |H|, \tag{4.6}$$

$$w_i^k \leq z_i \qquad \forall\, i \in V, k \in K, \tag{4.7}$$

$$\sum_{i\in V} w_i^k \leq 1 \qquad \forall\, k \in K, \tag{4.8}$$

$$x_{ij}^{kr} \leq \sum_{\ell\in V} w_\ell^k \qquad \forall\, i, j \in V, k \in K, r \in R, \tag{4.9}$$

$$\sum_{j\in V}\sum_{r\in R} x_{ij}^{kr} \geq w_i^k \qquad \forall\, i \in V, k \in K, \tag{4.10}$$

$$\sum_{\ell\in V}\sum_{r\in R} x_{\ell j}^{kr} \leq |V|\,(2 - w_i^k - z_j) \qquad \forall\, i, j \in V, j \neq i, k \in K, \tag{4.11}$$

$$\sum_{i\in V} w_i^{k+1} \leq \sum_{i\in V} w_i^k \qquad \forall\, k \in \{1,\ldots,|K|-1\}, \tag{4.12}$$

$$\sum_{j\in V} x_{ij}^{k,r+1} \leq \sum_{j\in V} x_{ij}^{kr} + 1 - z_i \qquad \forall\, i \in V, k \in K, r \in \{1,\ldots,|R|-1\}, \tag{4.13}$$

$$s_i + t_{ij} \leq s_j + 2\,b^d(1 - x_{ij}^{kr} + z_j + z_i) \qquad \forall\, i, j \in V, k \in K, r \in R, \tag{4.14}$$

$$h_1^{kr} + t_{ij} \leq s_j + 2\,b^d(2 - x_{ij}^{kr} + z_j - z_i) \qquad \forall\, i, j \in V, k \in K, r \in R, \tag{4.15}$$

$$s_i + t_{ij} \leq h_2^{kr} + 2\,b^d(2 - x_{ij}^{kr} - z_j + z_i) \qquad \forall\, i, j \in V, k \in K, r \in R, \tag{4.16}$$

$$h_2^{kr} \leq h_1^{k,r+1} \qquad \forall\, k \in K, r \in \{1,\ldots,|R|-1\}, \tag{4.17}$$

$$a^c \leq s_i \leq b^c \qquad \forall\, i \in V, \tag{4.18}$$

$$a^d \leq h_1^{kr} \qquad \forall\, k \in K, r \in R, \tag{4.19}$$

$$h_2^{kr} \leq b^d \qquad \forall\, k \in K, r \in R, \tag{4.20}$$

$$x_{ij}^{kr},\, z_i,\, w_i^k \in \{0,1\}\, s_i,\, h_1^{kr},\, h_2^{kr} \in \mathbb{R} \qquad \forall\, i, j \in V, k \in K, r \in R. \tag{4.21}$$

The objective function minimises three types of costs: the variable routing costs (fuel, vehicle maintenance, tolls, etc.), the fixed depot costs (depot lease/purchase and personnel wages) and the fixed vehicle costs (lease/purchase, insurance, driver wages, etc.).

Constraints (4.1) and (4.2) ensure that every node $i \in V$ is visited *exactly once* if it is not a depot and *at least once* otherwise. Constraints (4.3) stipulate that, if trip r of vehicle k enters node i, it must leave node i. We introduce constraints (4.4) to ensure that every vehicle leaves a depot at most once in each trip; they are necessary to avoid sub-trips within the same trip that could deceive the time window constraints by taking place simultaneously. The vehicle capacity constraints are defined in (4.5). If vertex i is the depot of vehicle k, then the demand of this node is excluded from the load. Constraints (4.6) ensure that exactly $|H|$ depots are placed. Constraints (4.7)-(4.8) assign vehicles to at most one depot. Constraints (4.9) ensure that if a vehicle is not used, this vehicle cannot traverse any arcs. Constraints (4.10) impose that a vehicle that does not complete any trip cannot be marked as used by variables w. This is needed to prevent negative demand on a node when evaluating the capacity constraints (4.5), which could happen if it is not costly to mark a vehicle as used.

We introduce constraints (4.11) to ensure that every vehicle in use always departs from its assigned depot by forbidding it to enter other depots. We also introduce symmetry breaking constraints (4.12) and (4.13) to impose a natural ordering in the selected vehicles and trips, respectively, and eliminate equivalent solutions. They ensure that vehicle $k+1$ cannot be used unless k is in use; the same is true for trips $r+1$ and r.

Constraints (4.14) - (4.20) are our adaptation of the time window constraints for the multi-depot multi-trip VRP. (4.14) are used when none of the nodes considered are a depot. (4.15) only applies when the first node is a depot, and (4.16) when the last node is a depot. Additional variables h_1^{kr}, h_2^{kr} are needed to define visit times to the depot, as there are two times associated with the depot for every vehicle and trip: start and end time of a trip. Because of this we do not need the vehicle index nor the trip index in s_i, as nodes that are not depots can only be visited once by one vehicle. Constraints (4.17) ensure that a trip r cannot start before the finish time of the previous trip completed by the same vehicle. Finally, constraints (4.18)-(4.20) enforce the time windows of customers and vehicles. Subtour elimination constraints are not necessary due to the presence of the time window constraints [69].

Using the formulation above, we can prove the complexity of the problem. By fixing the number of trips to one, the index r of any decision variables can be dropped. We can fix the depot at node 0 by forcing $z_0 = 1, |H| = 1$, and setting $a^d = a^c$ and $b^d = b^c$. With these changes, we notice that by removing the resulting redundant constraints, we obtain the VRPTW formulation. This shows that our problem is a *generalisation* of the VRPTW. Since the VRPTW is itself a generalisation of the TSP, which belongs to the class of *NP-hard* problems, we conclude that our problem is also *NP-hard* – there is no known algorithm that can solve this problem in polynomial time.

4.3 Methodology

In this problem there is a hierarchy of decisions that take place at different stages in the planning period:

1. *Strategic level.* Decisions involving capital expenditure (capex) that need to be made in advance and remain constant throughout the entire planning horizon; these are the location of depots and the fleet sizing.

2. *Operational level.* Decisions that are made on a daily basis and have an impact on operational expenditure (opex); these are the vehicle routes. Since customer demand is only known at 6pm the day before, these routes should be generated fast enough so that schedules can be communicated to drivers before they start their shift at 6am.

Because of this hierarchy, there is a business need to decompose the problem into separate sub-problems. We have developed a solution approach that addresses the different stages of the problem sequentially. First, the location of depots is optimized. Next, fleet sizes are determined for each depot. After these strategic decisions have been fixed, we address the operational planning aspect: on a given day, customers that require a visit are assigned to depots and vehicle routes are optimized.

All our solution methods are implemented in AIMMS, with CPLEX as the MIP solver.

4.4 Minimum number of depots

Since the number of depots is a parameter and not a decision variable, we are interested in identifying the minimum number of depots that preserves feasibility. This number can be determined by solving a *Set Covering Problem* (SCP). The SCP aims to place facilities at locations such that every node can be reached from at least one facility. Each depot can be considered to have a radius of coverage equal to the longest distance that a vehicle could cover in a single trip without violating the time windows.

In our problem, the radius of coverage is d_{max} = 5.5 h × 40 mph = 220 miles. A vehicle could visit a customer located at most 5.5 hours away from the depot, although this would likely result in a large fleet size with a poor vehicle utilisation. Given the data, the solution to the SCP is trivial, as there are 10 ZIP codes such that the location of all customers is less than 5.5 hours away from any of these 10 locations. Therefore, one depot suffices to satisfy all the demand in 2018, provided there are enough vehicles to complete the generated trips.

4.5 Depot location

To optimize the location of a given number of depots without solving the routings for every day in the 2018 data, which would require very costly computations, we treat this as a facility location problem. We are interested in finding the best locations of a

given number of depots such that, if all customers are assigned to their closest depot, the weighted average distance between customers and their assigned depots will be minimum. The rationale is that minimising this metric should result in shorter vehicle trips.

To obtain the weights for the average distance, we aggregate demand in 2018 by the number of days that each customer has positive demand. The more often a customer requires a visit, the more it will drive the location of the nearest depot towards it. We observe a strong correlation between this metric and the total annual demand of a customer in our data (the Pearson coefficient between the two metrics is $\rho = 0.998$). Therefore, choosing one metric or the other is not expected to have a significant impact on the outcome of the model.

In addition, the fixed costs of depots are also included in the objective function. Since we cannot estimate routing and vehicle costs from the average customer-depot distances, it is not straightforward to find the right balance between the depot costs and the average customer-depot distance. To overcome this difficulty, we include both terms in the objective function with a parameter $\alpha \in [0, 1]$ that controls their relative importance and we choose the best value for α by inspection.

Our formulation is a variation of the p-median problem [33]. Variables z_i indicate whether a depot is placed at location $i \in V$ and variables y_{ij} indicate whether node i is assigned to a depot located in j. d_{ij} is the distance between nodes i and j, $d_{\max}$ is the maximum distance allowed between a demand node and a depot due to the time window constraints, and Ω_i is the number of days that customer $i \in V$ had positive demand in 2018, namely $\Omega_i = \sum_d 1_{(q_{id}>0)}$.

$$\min \quad \frac{\alpha}{\sum_{i\in V}\Omega_i} \sum_{i\in V}\sum_{j\in V} \Omega_i\, y_{ij}\, d_{ij} + (1-\alpha) \sum_{j\in V} f_j\, z_j \tag{4.22}$$

$$\text{s.t.} \quad \sum_{j\in V} y_{ij} = 1 \qquad \forall\, i \in V, \tag{4.23}$$

$$\sum_{j\in V} z_j = |H|, \tag{4.24}$$

$$y_{ij} \le z_j \qquad \forall\, i, j \in V, \tag{4.25}$$

$$d_{ij}\, y_{ij} \le d_{\max}\, z_j \qquad \forall\, i, j \in V, \tag{4.26}$$

$$y_{ij},\, z_j \in \{0, 1\} \qquad \forall\, i, j \in V. \tag{4.27}$$

Constraints (4.23) ensure each node is assigned to exactly one depot. (4.24) is the p-median constraint, where the p parameter is represented by the cardinality of the set of depots, $|H|$. Constraints (4.25) stipulate that nodes can only be assigned to depots, not to customer nodes. The main difference with the p-median problem lies in constraints (4.26), which are needed to ensure that only nodes that can be *reached* from a depot can be assigned to that depot.

This p-median problem, with circa 1,500 nodes or ZIP codes, is too large to solve to optimality with exact methods. Our approach consists of solving the p-median problem in two stages, considering different techniques:

- *Stage 1*: Group nodes that are close together into clusters, forming at most 500 clusters. Then solve the p-median problem using these clusters instead of the original

nodes to identify $|H|$ clusters that contain suitable depot locations. Since each of these $|H|$ clusters will generally include multiple nodes (multiple ZIP codes), Stage 2 assigns a depot to a single node in each of the clusters.

- *Stage 2*: Consider the p-median problem with the original nodes. Fix $z_j = 0$ for all nodes $j \in V$ that do not belong to the clusters identified in Stage 1. This ensures that only nodes that are in a cluster are candidate depot locations. Finally, solve the resulting p-median problem to obtain the final set of depot locations H.

Both steps run in less than a minute, while introducing only a small distortion to the spatial distribution of demand. To perform the clustering described in Stage 1, we apply k-medoids [71], an unsupervised machine learning technique similar to the well-known k-means algorithm [94]. For a given number of clusters k, k -medoids minimises the dissimilarities between points in a cluster and the centre of that cluster. It can be easily implemented if the input data is a distance matrix. The code to perform the clustering of nodes was written in Python using the library PyClustering [105].

A visualisation of the trade-off between optimizing for minimum cost of locating depots and minimum average distance customer-depot is given in Figure 4.1. Optimal solutions are obtained for combinations of 11 different, evenly spaced, values of $\alpha \in [0,1]$ and four representative numbers of depots. It can be observed that, as we prioritise minimising the total fixed cost of depots ($\alpha \to 0$), the average distance depot-customer grows very rapidly. Similarly, if only the average distance metric is considered ($\alpha = 1$), then solutions with very high depot costs are obtained. However, by reducing α to 0.9, namely, by adding a small penalty for high depot costs, these costs are reduced between 20% and 40% while the increase in the average distance is negligible. This suggests that it is easy to find a cheaper alternative in the neighbourhood of a location that is optimal under spatial considerations only.

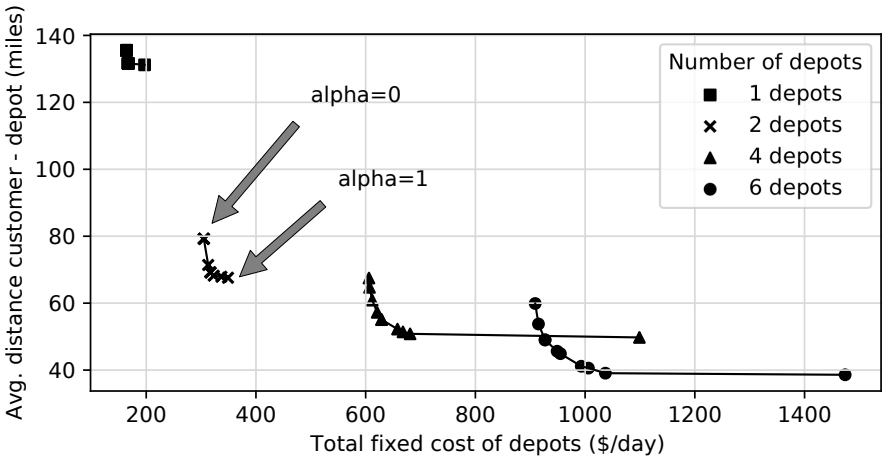


Figure 4.1: Cost of locating depots vs average distance customer-depot

Unlike routing costs, which are 100% opex, the fixed cost of the depots has a capex element (lease or purchase payments). This means that part of the depot costs will be amortised within a few years, whereas routing costs are running costs. Therefore, it seems sensible to invest in finding good locations for the depots at the expense of

purchasing or leasing slightly more expensive facilities. For these reasons, we set α to 0.9. Using 2018 data, this scheme returned a depot configuration in 40 seconds on average for different numbers of depots.

It should be noted that balancing of demand across different depots has not been included in this model. Having some variability in the volume of demand assigned to each depot is not considered to be problematic since it is possible to assign fleets of different size to different depots.

4.6 Fleet sizing

The next strategic decision concerns the sizing of the fleet and the distribution of this fleet across different depots. This aspect of the problem is key to obtaining robust solutions. While a poor choice of depot locations or number of depots can be compensated with a larger fleet of vehicles, the fleet sizing is critical to ensuring that all demand can be routed: too small a fleet will result in some customers not being visited on certain days unless additional resources are put in place.

To obtain a fleet size that can cope with 2018 demand but also be robust against unseen demand data (e.g. 2019 demand), it is not sufficient to set the fleet size to the number of vehicles required on the busiest day in 2018, because the fleet size required depends not only on the total demand but also on the spatial distribution of the customers. To overcome this, we propose to simulate a set of demand instances based on a busy period in 2018. We then determine the number of vehicles required on those simulated days by solving a VRP on each of these instances (using the methodology outlined in Section 4.8) and decide the fleet size based on these results.

Simulating “busy days" first requires an understanding of the demand distribution in 2018. Let $q(i,d)$ be the demand of customer i in day d. We assume that $q(i,d)$ is given by the product of two random variables $X(i,d)$ and $Y(i,d)$. $X(i,d)$ is a binary variable indicating whether customer i is visited on day d and $Y(i,d)$ indicates how many demand units customer i must be served on day d. The rationale behind this assumption is that we observed that, on average, only 5% of the customers had a positive demand ($q(i,j) > 0$) on any given day in 2018, and in those cases where $q(i,j) > 0$, this demand was variable and seemed to follow a distribution centred in 10.

Through statistical analysis we identified the following properties of X and Y that can be exploited to simulate instances that resemble the actual demand in 2018:

1. *The amount of goods required by an active customer node, $Y(i,d)$, is independent of the customer and the day.* $Y(i,d)$ can be replaced with Y.

2. *The amount of goods required by an active customer node, Y, follows a Poisson distribution with $\lambda = 9.95$.* The parameter λ was fitted using the maximum likelihood method. The goodness of fit was tested using a Chi-Squared test, and it can also be visualised in Figure 4.2.

3. *$X(i,d)$ is a Bernouilli distributed random variable* that takes value 1 if customer i requires a visit on day d, which happens with probability p_{id}. This probability is different for every customer and day (see next point).

4. *The seasonality behind the demand in 2018 is driven by $X(i,d)$.* The R^2 score of a linear regression model linking the number of active customers with the total demand on a given day is 0.986, which indicates that 98.6% of the variability in the total demand is explained by the number of active customers. Busier days are so because there are more customers that have to be visited, and not because the demand per active customer is higher. The similarity between these two time series can be seen in Figure 4.3.

5. With the exception of a few days in January, the *busiest period of the year is the summer season* (see Figure 4.3). Also, a logistic regression model showed that *customers are 20.3% less likely to have demand on a weekend.*

6. *On any given day, the demands of different customers in terms of $X(i,d)$ are independent from each other.* The Pearson correlation coefficients ρ that resulted from comparing the demands of all pairs of customers are very low ($|\rho| < 0.15$ for 95% of all pairs of customers and $\max|\rho| = 0.26$). As a result, we can expect that the spatial distribution of demand will be similar on different days. In other words, the relative volume of customers in each depot area will not vary significantly on different days. This is a convenient property from a fleet sizing point of view given that vehicles must operate from the same depot every day.

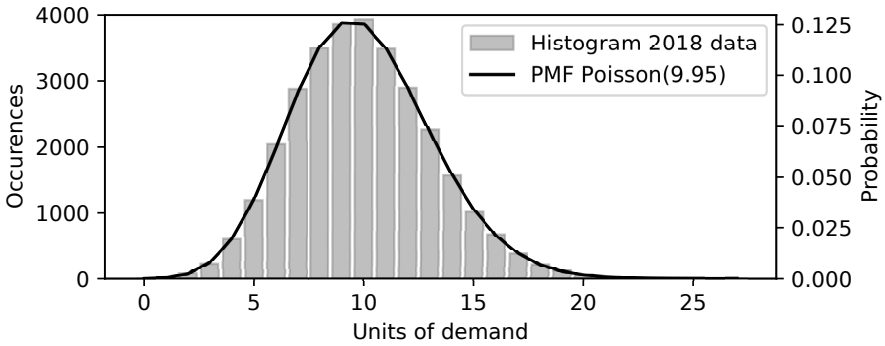


Figure 4.2: Distribution of Y in 2018 and fitted Poisson distribution

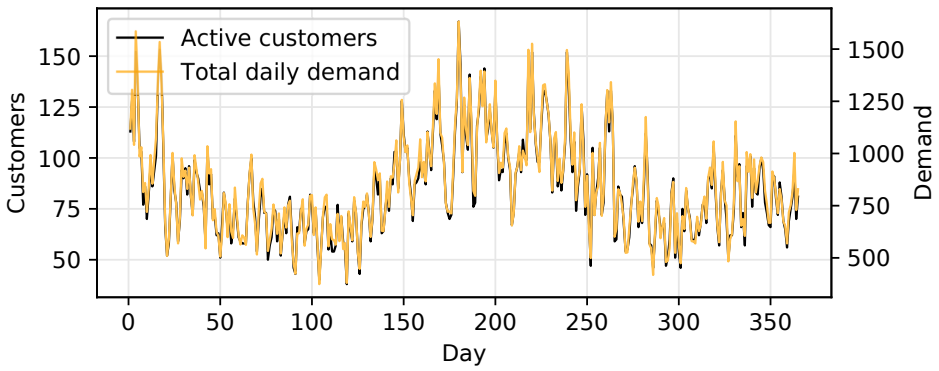


Figure 4.3: Total demand and number of active customers per day in 2018

Based on these observations, the following approach is used to simulate a “busy day":

1. *Sample from X*: each customer i is activated with a probability $p_{i,busyday}$. This probability is initially estimated as the frequency with which customer i required a visit over all weekdays from June to August in 2018. In other words, we sample data from an artificial “busy day" that behaves like an average summer weekday in terms of demand volume, and the probability $p_{i,busyday}$ will only depend on the customer node i. Furthermore, to achieve more robust solutions, customers that were not active during this period (and therefore would have zero chance of appearing in our simulated data) are assigned a probability p equal to the lowest non-zero probability in the data.

2. *Sample from Y*: for each customer that has been activated in the previous step, the amount of demand is sampled from $Poisson(9.95)$.

3. *Calibrate demand level*: to obtain robust solutions that can ensure not only that all demand is satisfied in 2018, but also cope with a possible increase in demand in the future, we introduce a calibration factor $\beta > 1$ that increases the generated demand by multiplying $p_{i,busyday}$ by this factor at the time of simulating data. We choose to increase $p_{i,busyday}$ and not the parameter λ because it is variable X, and not Y, that is affected by seasonality.

In summary, the demand of customer i on a simulated “busy day" d is obtained from equations (4.28)-(4.30):

$$X \sim \text{Bernouilli}(p_{i,busyday} \times \beta) \tag{4.28}$$

$$Y \sim \text{Poisson}(9.95) \tag{4.29}$$

$$q(i,d) = X \times Y \tag{4.30}$$

The value of β plays a crucial role in ensuring the robustness of our solutions; it will determine by how much we underestimate or overestimate the fleet size required. We tune this parameter by inspection to ensure that the average demand of the simulated instances is approximately equal to some demand level that we define in advance. In particular, we have explored several demand levels that may be of interest to the business: the maximum in 2018, the 95$^{\text{th}}$ percentile (P95) and 75$^{\text{th}}$ percentile (P75).

Demand level	Active customers	Calibrated β
Max demand 2018 + 10%	184	1.7
Max demand 2018	167	1.55
P95 demand 2018	133	1.25
P75 demand 2018	115	1.05

Table 4.1: Fleet sizing results for 4 demand levels

Finally, we solve the fleet sizing problem by completing the following steps:

Step 1 Choose a design criterion with its associated β from Table 4.1.

Step 2 Simulate n demand realisations with (4.28)-(4.30).

Step 3 For each generated instance, solve a VRP and store the number of vehicles required at each depot.

Step 4 Set the final fleet size assigned to each depot to the 95th percentile of the list of n vehicle requirements obtained in Step 3. This is different from the P95 used for simulating demand.

We choose to set the fleet size assigned to each depot to the 95th percentile of the list of n vehicle requirements to be conservative: it is sufficient to route the demand of 95% of the generated instances and it is less sensitive to extreme values in the results than the maximum requirement. We set the number of simulations n to 50, which is high enough to obtain a stable number of vehicles with the 95th percentile, but not too high that it takes a prohibitive amount of time to run the fleet sizing model. Lastly, we solve these 50 routing instances using our heuristic algorithm described in Section 4.8 with the standard savings function and no route improvements, all of which speed up the process and lead to more pessimistic and conservative results.

Results of running the fleet sizing algorithm for the four proposed demand levels in Table 4.1 are displayed in Figure 4.4. They show the estimated values of β to generate a set of instances with an average demand (in number of customers) approximately equal to 4 different levels of demand from the 2018 data. To simplify the analysis, we ran the algorithm for a single representative number of depots (four depots). The horizontal lines represent the fleet size obtained using different demand levels, whereas the time series represent the actual vehicle requirements on any day in 2018 and 2019.

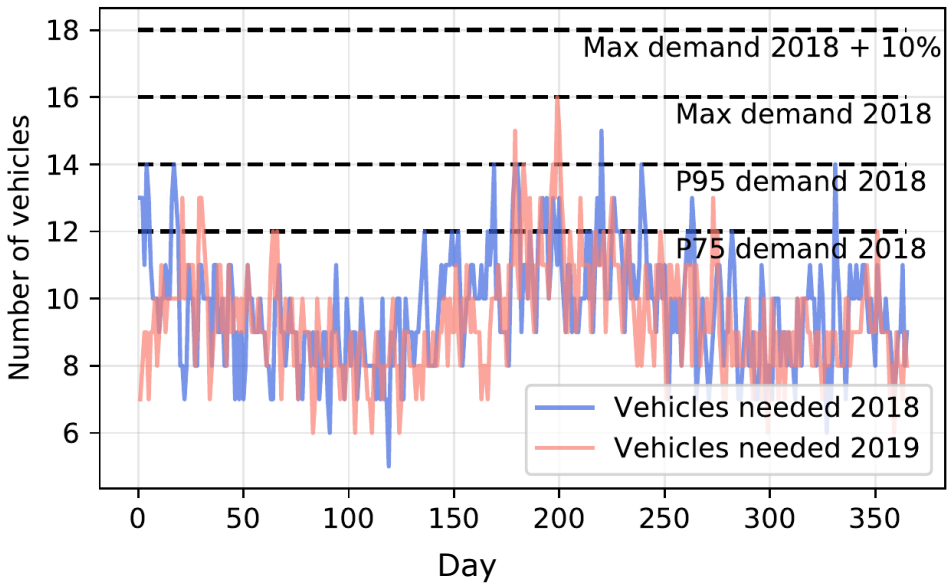


Figure 4.4: Minimum number of vehicles required for 2018 and 2019

We can see that the same seasonality pattern is present in 2019 and there does not seem to be a positive trend in the year-over-year growth, although the maximum fleet requirement in 2019 (16 vehicles) was one unit higher than the highest requirement in 2018 (15 vehicles).

When the simulated demand is calibrated using the maximum demand in 2018, the resulting fleet size is sufficient to route all demand in 2018 and 2019. The minimum number of vehicles required to serve the demand everyday in 2018 and 2019, compared to the fleet size calculated using different design criteria is shown in Figure 4.4. All the VRP instances considered in this analysis have been solved using the heuristic method from Section 4.8 to speed up the solution.. However, if the decision maker is risk-averse and does not have any contingency measures for when the existing fleet cannot cope with the upcoming demand, they should use the “maximum demand 2018 + 10%" criterion, which, in this case, gives two additional vehicles for safeguarding.

On the other hand, the P75 and P95 design criteria yielded fleet sizes that did not meet the vehicle demand in 2018 nor 2019. However, it is worth noting that the maximum demand in 2018 was significantly higher than average levels throughout the year: the maximum number of customers was 25% higher than the 95^{th} percentile and 45% higher than the 75^{th} percentile. This shows that designing for the peak will result in a low utilisation of vehicles for most of the year. Therefore, if the business had the ability to rent additional vehicles for a day on short notice, there would be scope for potential savings by designing the fleet using the the 75^{th} percentile criterion. For the case presented in Figure 4.4, this resulted in a fleet of 12 vehicles, which suffices to serve the demand of most days.

To estimate the impact on costs of these different approaches we present in Table 4.2 an analysis of the additional rental vehicles that would be required in 2018 and 2019 using different calibration criteria. We are comparing the fleet size, fleet fixed costs, additional vehicles required when the existing fleet is insufficient to meet demand, associated costs of these extra vehicles and total vehicle costs, for each year. To be conservative we have assumed that the daily rate to rent a vehicle on short notice is triple that of standard vehicles, namely, \$90/day.

It is clear from these results that designing the fleet for peak demand results in a low utilisation of vehicles for most of the year. If it is possible to rent vehicles on an ad-hoc basis, even under our conservative assumption on rental costs, the cost of doing so using the P75 and P95 demand 2018 criteria would be negligible compared to the fixed vehicle costs.

Our approach to robust fleet sizing has the additional advantage that it allows the decision maker to manage risk and costs effectively. These criteria are intuitive and do not involve tuning a mysterious numeric parameter that is hard to interpret; β is updated in the background. It took between 25 ("P75 demand") and 90 seconds ("Max demand + 10%") to run this fleet sizing model in a four-depot scenario.

4.7 Assignment of customers to depots

We now enter the operational planning stage of the problem, which aims to determine the routes to be completed by each vehicle in each trip such that all the demand of a

Design criterion	P75	P95	Max	Max + 10%
		2018		
Vehicles assigned	12	14	16	18
Fixed vehicle costs	\$131k	\$153k	\$175k	\$197k
Extra vehicles needed	32	1	0	0
Cost of extra vehicles	\$2.9k	\$90	\$0	\$0
Total vehicle costs	\$134k	\$153k	\$175k	\$197k
		2019		
Vehicles assigned	12	14	16	18
Fixed vehicle costs	\$131k	\$153k	\$175k	\$197k
Extra vehicles needed	26	4	0	0
Cost of extra vehicles	\$2.3k	\$360	\$0	\$0
Total vehicle costs	\$134k	\$154k	\$175k	\$197k

Table 4.2: Comparison between different design criteria

given day is served.

The mathematical model presented in Section 4.2 would simultaneously assign customers to depots and determine the optimal routes. However, this would require to solve a large multi-depot VRP. To overcome this, we assign customers to depots *a priori*. This allows us to treat the routing problem as a set of independent, smaller VRP's, one for each depot.

A reasonable approach to achieve this would be to assign customers to their closest depot for any demand realisation. However, this does not take into account the spatial distribution of the demand on a given day, e.g., it might be "easier" to visit a customer by a vehicle completing a trip nearby, even if this vehicle is based on a depot that is not the closest to the customer.

For this reason, in this section we explore the effect of factoring in the "routing effort" required to visit a customer. Jarrah and Bard [68] designed a metric based on the intra-customer travel time to address the problem of geographic clustering of customers. They estimate the travel time from customer i to the next customer based on the distance to the k nearest customers in the same cluster.

We have explored this idea of the distance to the k nearest customers as a proxy of the routing effort in the problem of assigning customers to depots on a given day. The approach we propose is as follows: starting from an initial assignment of customers to their closest depots, we iterate over the set of customers and we assign each customer $i \in V$ to a depot $\ell \in H$ such that the average distance from i to the k nearest neighbours currently assigned to depot ℓ plus the distance from i to the depot is minimum. This metric is

$$\text{depot of customer } i \equiv \ell^* = \arg\min_{\ell} \left\{ d_{\ell i} + \frac{1}{k} \sum_{j \in V_\ell^k} d_{ij} \right\}, \tag{4.31}$$

where V_ℓ^k is the set k customers currently assigned to depot ℓ that are the closest to customer i. Because every time we change the depot assigned to a customer this might change the k nearest neighbours of customers that we have already scanned,

we repeat this process iteratively until the assignments are stable, i.e., until there is no change in the assignment of any customer in one round.

Including the distance to the depot in equation (4.31) is necessary to ensure that, in a stable solution, the clusters of customers are still centred around the depots. Without this term, the iterative assignment process could converge in $|H|$ clusters of customers with no connection with the depot locations. This was not considered in [68], since this study was only concerned with cluster construction.

It only remains to choose a value of k. Figure 4.5 shows the impact on routing costs of including the average distance to the k nearest neighbours when assigning customers to depots. The results for the different values of k represent the average routing costs from 50 randomly selected instances with different numbers of depots. Including the nearest neighbours in the assignment always results in savings in routing costs for the values of k tested in this experiment. With $k = 2$, there is a cost reduction of circa \$45/day with respect to assigning customers to the closest depot ($k = 0$). Although this value is small compared to the daily routing costs, it adds up to an annual \$18k saved on routing costs and it can be achieved with very little computational effort.

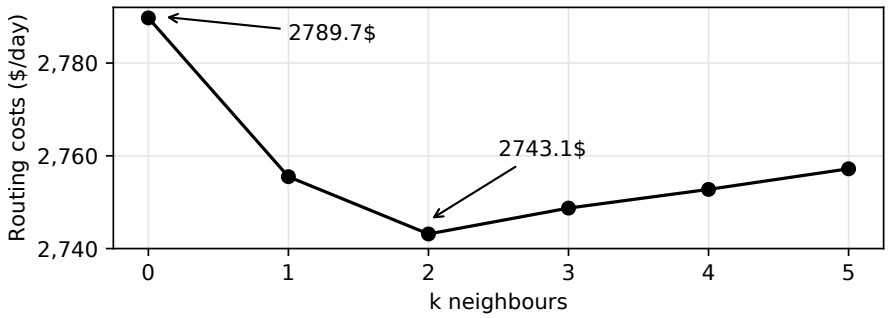


Figure 4.5: Analysis on the number of nearest neighbours

4.8 Vehicle routing

At this point, the original routing problem has been divided into $|H|$ sub-problems (one for each depot) of a more manageable size. These sub-problems belong to the class of *Multi-Trip Vehicle Routing Problems with Time Windows* (MTVRPTW) [9, 14]. There is now a single depot in each sub-problem and a known fleet size denoted by $|K|$. The mathematical model in Section 4.2 can be employed to solve some of these instances, with some considerations:

1. Decision variables z_i, which determine the location of depots, are now known and can be treated as problem data ($z_i = 1$ if customer i is the depot of a given sub-problem, 0 otherwise).

2. Constraint (4.6), which fixes the number of depots to $|H|$, is no longer needed.

3. Constraints (4.11), which ensure that the same vehicle cannot complete two trips in two different depots, can also be removed.

4. We set the cardinality of the set of vehicles K to the number of vehicles assigned to the depot to ensure the number of vehicles used does not exceed the fleet size. Even though the daily fleet costs are fixed regardless of the daily vehicle utilisation, we keep them in the objective function to encourage solutions with high fleet utilisation. This avoids solutions where many or all the available vehicles are assigned to one or two short trips, which would not be desirable from a driver rostering perspective. Given the low cost of the vehicles, it is safe to add fleet costs in the objective function without significantly affecting the routing costs, which are the main cost component to minimise at this stage.

The computational experiments carried out to test the limits of this mathematical model, presented in Section 4.8.3 below, revealed the need for a more sophisticated exact method for smaller instances and a heuristic approach that can handle the larger instances in reasonable time at the expense of generating sub-optimal routings in some cases. Based on these results, we adopt a hybrid approach: on any day, instances (or depots) with fewer than 15 customers are solved using the branch-and-cut algorithm described in Section 4.8.1 with a time limit of 20 minutes; for larger instances we resort to the heuristic method described in Section 4.8.2. This time limit of 20 minutes is high enough to obtain acceptable solutions for small instances but low enough that routes can be generated in an acceptable amount of time after the demand data becomes available at 6pm.

4.8.1 Exact Method

In an effort to improve the efficiency of the exact method implemented by CPLEX, we propose a VRP-specific branch-and-cut approach. For this purpose, we define a new set of valid inequalities that are added sequentially in the form of cuts in the branching tree. We use the dual simplex method to solve the linear programming (LP) relaxation of the original MIP problem at every node, disable all of the built-in cuts used by CPLEX, apply a coefficient reduction technique and use our heuristic solution to warm start the MIP.

We ran a series of experiments to decide on the most appropriate method to apply the cuts throughout our branching tree, while balancing the time needed for separation, the number of additional constraints and the rate of improvement. As a result of our experiments, we perform a separation procedure in 10 consecutive nodes in the branching tree, remove all possible violations, and repeat this separation once in every 500 inspected nodes. This way, when the algorithm enters the 1500^{th} node, only 30 nodes will have been inspected for violations, and the risk of adding an overwhelming number of cuts that may slow down the solution process is lowered. All the cuts added are globally valid, and the results obtained are presented in Section 4.8.3.

Valid Inequalities

If the depot nodes are excluded from the graph of our problem, a feasible solution should only consist of *open paths*. Therefore any cycle in this graph represents a violation of a constraint of the problem and should be eliminated. This logic leads

to the most common valid inequalities for the TSP, known as subtour elimination constraints. Even though in our case the time window constraints (4.14)-(4.20) prevent subtours in any integer solution, the fractional solutions resulting from solving the linear relaxation of the problem can easily contain subtours. By implementing an adequate separation procedure, one is able to identify these violations and apply an appropriate constraint that is globally valid, thus making this solution infeasible in all future nodes of the branch-and-bound tree. The constraints known as *lifted subtour inequalities* ([44]) are facet defining for the TSP for $|S| \geq 3$, $S \subseteq V$. These inequalities have shown good results when applied to the VRPTW [13] and will be used to tighten the polyhedron of the LP relaxation of our problem. Due to the form of our degree constraints (4.1) and (4.2) and the fact that we are allowing multiple trips per vehicle, we need to implement the lifted subtour inequalities in a different form:

$$\sum_{\substack{k\in K\\ r\in R}} \sum_{j=1}^{|S|-1} x^{kr}_{i_j i_{j+1}} + \sum_{\substack{k\in K\\ r\in R}} x^{kr}_{i_{|S|} i_1} + 2\sum_{\substack{k\in K\\ r\in R}} \sum_{j=2}^{|S|-1} x^{kr}_{i_j i_1} + \sum_{\substack{k\in K\\ r\in R}} \sum_{j=3}^{|S|-1} \sum_{h=2}^{j-1} x^{kr}_{i_j i_h} \leq |S|-1, \quad S \subseteq N, |S| \geq 3, \tag{4.32}$$

$$\sum_{\substack{k\in K\\ r\in R}} \sum_{j=1}^{|S|-1} x^{kr}_{i_j i_{j+1}} + \sum_{\substack{k\in K\\ r\in R}} x^{kr}_{i_{|S|} i_1} + 2\sum_{\substack{k\in K\\ r\in R}} \sum_{j=3}^{|S|} x^{kr}_{i_j i_1} + \sum_{\substack{k\in K\\ r\in R}} \sum_{j=4}^{|S|} \sum_{h=3}^{j-1} x^{kr}_{i_j i_h} \leq |S|-1, \quad S \subseteq N, |S| \geq 3. \tag{4.33}$$

The subtours of size $S = \{i, j\}$ ($|S| = 2$) will be cut off using

$$\sum_{\substack{k\in K\\ r\in R}} x^{kr}_{ij} + \sum_{\substack{k\in K\\ r\in R}} x^{kr}_{ji} \leq 1. \tag{4.34}$$

Separation Procedure

Once a feasible solution for the LP relaxation has been found, the edges from set E in the graph $G(N, E)$, where V is the set of original customers, are assigned a weight $w(i, j) = \sum_{k\in K} \sum_{r\in R} x^{kr}_{ij}$. To separate the violated subtour inequalities (4.32), (4.33) and (4.34) we use a shrinking procedure. A similar procedure was introduced in Section 1.6 and used in the branch-and-cut algorithm in Section 2.8 (see Algorithm 2.5).

The separation procedure consists of creating additional nodes in the graph, while simultaneously removing both end nodes of an edge with positive weight. Each new node created in the graph is a *child* node, and the nodes that are removed during this process are its *predecessors*. That way, each child node has a line of predecessors that can be backtracked to the set of original nodes that generated it. Let $n = |N|$ and assume a feasible solution has been obtained for the LP relaxation. The procedure is as follows:

Step 0 For each node $i \in V$ initiate an empty set of predecessors.

Step 1 Find an edge (i, j) such that $w(i, j) > 0.5$ and $w(j, i) > 0.5$. If such edge exists, go to Step 5. Otherwise, go to Step 2.

Step 2 Find an edge (i, j) such that $w(i, j) > 0$. If such edge exists, go to Step 3. Else, **Stop**.

Step 3 Create child node $\ell := n+1$, and update the weights $w(s, \ell) := w(s, i) + w(s, j)$ and $w(\ell, s) := w(i, s) + w(j, s)$, for every $s \in N, s \neq i, j$. Update $n := n+1$.

Step 4 Remove ℓ's predecessors i and j from N ($N := N \backslash \{i, j\}$). Update the set of ℓ's predecessors with nodes i and j and all of their own predecessors. Go to Step 1.

Step 5 Create a set S of original nodes containing i and j with their predecessors. If $|S| = 2$, insert cut (4.34). If $|S| > 2$, insert both cuts (4.32) and (4.33). Re-optimise.

4.8.2 Routing heuristic

For instances with more than 15 customer nodes we have developed a heuristic algorithm. This heuristic has three stages: first, the multi-trip characteristic is ignored and routes are constructed considering only vehicle capacities and time windows. Next, solution improvement techniques are employed to improve the routes constructed in the previous step. Lastly, a packing algorithm assigns routes to trips of vehicles.

Route construction heuristic

Our route construction approach is inspired by the adaptive routing heuristic proposed by Batarra et al. in [14]. The authors first determine the period of time during which each route is strongly active. A route is strongly active at some time interval if it is guaranteed that the route will be in progress at that time, regardless of when it starts, see Figure 4.6.

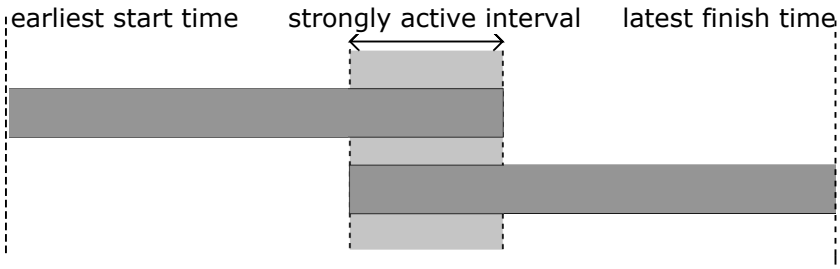


Figure 4.6: Illustration of a strongly active interval of a route

Since the number of routes that are strongly active at the same time define a lower bound on the number of vehicles required, Batarra et al. [14] penalise routes that are strongly active at the busiest time, the *critical time*, and adjust the insertion costs in their route construction heuristic to include these penalties. The expected result is that routes constructed using these penalties will avoid the critical times and will therefore be easier to pack in the packing stage of the algorithm. They repeat this

two-stage process iteratively (constructing routes - calculating penalties) until some stopping condition is met.

In the VRPTW considered in this work, all customers can be visited between 8am and 4pm and therefore the critical time will always occur around the centre of this time window. This is not the case in [14], where different customers can have different time windows and therefore critical times may occur at any time of the day. We take advantage of this property of our problem to introduce two substantial simplifications to the approach in [14]: first, the penalty for the critical time interval is a fixed value that we set in advance; second, routes are only constructed once, including the penalties.

We propose a parametric savings heuristic based on the general parallel savings heuristic developed by Clarke and Wright [27]. We define the savings function as follows:

$$s(i,j) = s_{routingtime}(i,j) + \lambda\, s_{stronglyactive}(i,j). \tag{4.35}$$

The first term is the well-known savings function, namely, $s_{routingtime}(i,j) = t_{0i} + t_{j0} - t_{ij}$. It represents the savings in travelling time when routing a vehicle from depot 0 to blackcustomer i to customer j and back to the depot as opposed to visiting i and j alone in different trips. We introduce a second term, $s_{stronglyactive}(i,j)$, which represents the benefit of joining route finishing in customer i, with route starting with customer j in terms of how “easy" the resulting route would be to pack compared to how “easy" it would be to pack the individual routes. $s_{stronglyactive}(i,j)$ is defined in equation (4.36):

$$\begin{aligned} s_{stronglyactive}(i,j) = f_{correctedSA}(\text{route with } i \text{ and } j) - \\ f_{correctedSA}(\text{route of } i) - f_{correctedSA}(\text{route of } j). \end{aligned} \tag{4.36}$$

The term $f_{correctedSA}$ represents a function that penalises routes that are strongly active at some time. We introduce here an additional modification with respect to the approach in [14]. We seek to penalise the amount of strongly active time only up to a certain point. As the duration of a route starts to approach the vehicle time window length, i.e., 11h, it becomes more convenient to keep expanding this route, because the remaining free time of the vehicle would become more and more difficult to fill by another route. In other words, it is the routes of medium duration which can make the packing of routes in vehicles inefficient. Using this logic, we define $f_{correctedSA}$ as follows:

$$f_{correctedSA} = \min\{11\text{ h} - f_{SA},\, f_{SA}\} \tag{4.37}$$

where f_{SA} is simply the amount of time that a route is strongly active. Based on this definition, $f_{correctedSA} = 0$ when a route is not strongly active at any time (f_{SA}=0), it increases linearly reaching a maximum when f_{SA}=5.5h, at which point it decreases until f_{SA}=11h, where $f_{correctedSA}$ is equal to 0 again. This means that no penalty is applied to routes that are not strongly active at any time nor routes that are 11h long.

Lastly, the term λ in (4.35) controls the weight of the term $s_{stronglyactive}(i,j)$ in the savings function and it has been set to 0.05 through computational experiments.

The rest of the algorithm follows the same structure as the parallel savings algo-

rithm by Clarke and Wright [27]. Pairs of customers (links) are sorted in decreasing order by their savings. Starting from the first link on the list, links are used to join smaller routes into larger routes as long as they are feasible in terms of vehicle capacity and time window restrictions. The algorithm stops when the list contains no more feasible links.

Including the term $s_{stronglyactive}$ to the savings function does add a small amount of computational time, as this term varies with the duration of the routes, and the savings list needs to be updated multiple times. Nevertheless, the total time in the worst case scenario, a single depot on a “busy day", was under six seconds.

Solution improvement

After constructing the initial routes, two solution improvement techniques are employed to possibly improve the current routes by reducing the routing costs. First, a TSPTW is solved to optimality for each of the constructed routes to ensure customers are visited in the shortest possible path within the route. For the sake of brevity, the MILP formulation of this TSPTW is not presented here. Last, we apply an exchange heuristic where we swap two customers visited in different routes with the aim to reduce the routing distances further (Section 1.7).

Packing heuristic

The heuristic we propose is a modification of the well-known *First Fit Decreasing* (FFD) heuristic [40], which sorts items in descending order of size, takes one item at a time and either assigns it to the first bin where it fits or opens a new bin if it did not fit in any of the open bins.

We propose a variation of the FFD heuristic in which the items (the routes) are also sorted in decreasing order by their size (route duration) and they are assigned to the first vehicle in which they fit, at the earliest possible start time, such that the time windows are not violated. The main difference with the FFD is that, when we assign a route to one trip to a vehicle, we pay attention to *where* in the current schedule of that vehicle it is inserted.

We will insert the route at the start of the schedule if doing so reduces the difference between the depot opening time and the earliest start time of the first trip currently scheduled on that vehicle. If that is not the case, we insert it at the end if the route has a longer travel time from the last customer to the depot than the current last trip. Otherwise, we insert it in any intermediate position. The rationale for this modification of the FFD algorithm is that we want to maximise the use of the two-hour and one-hour gaps at the start and end of the day, respectively, where the depot is open but customers cannot be served.

Furthermore, we observe that most routes can be operated in either direction, namely, customers can be visited in reversed order. This is true because the time windows of all customers are identical and the graph is undirected. The only case in which a route cannot be reversed is illustrated in Figure 4.7b. In light of this observation, we allow routes to be inserted in reverse order if it is feasible and beneficial.

Algorithm 4.1 Modified FFD heuristic

procedure PACKROUTESINVEHICLES($routes$, a_D, b_D, a_C, b_C)
  initialise list of $vehicles$ with one empty vehicle $k = 1$ and assign $k.gap \leftarrow a_C - a_D$ /* if depot opens at 6am and customers can be visited from 8am, the initial gap is 2h
  Sort $routes$ in decreasing order by route duration
  for r in $routes$ **do**
    for k in $vehicles$ **do**
      $r \leftarrow$ LONGARCFIRST(r) /* see procedure below
      if $(r.earliest - a_D) \leq k.gap$ **then** /* if placing route r at the start would reduce current starting gap
        Insert route r at the start of schedule of vehicle k
        if ISFEASIBLESCHEDULE(k, a_D, b_D, a_C, b_C) **then** /* see procedure below
          $k.gap \leftarrow r.earliest - a_D$ /* update current start gap
          break /* this route has been assigned, exit vehicle loop and move to next route
        else
          Remove route r from vehicle k
          continue /* scan next vehicle, this route cannot fit here
      else /* if this route cannot improve the starting gap
        $r \leftarrow$ LONGARCLAST(r) /* see procedure below
        if last arc in r longer than last arc of current route in last position **then**
          Insert route r at the end of schedule of vehicle k
        else
          Insert route r in any intermediate position in vehicle k
        if ISFEASIBLESCHEDULE(k, a_D, b_D, a_C, b_C) **then**
          break /* this route has been assigned, exit vehicle loop and move to next route
        else
          Remove route r from vehicle k
    if r has not been assigned **then**
      Insert vehicle at the end of list $vehicles$
      $r \leftarrow$ LONGARCFIRST(r)
      Insert route r at the start of schedule of vehicle k
      $k.gap \leftarrow r.earliest - a_D$ /* initialise start gap
  return number of elements in list $vehicles$

procedure LONGARCFIRST(r)
  Arrange route r in a direction such that it starts with larger between $r.first$ and $r.last$

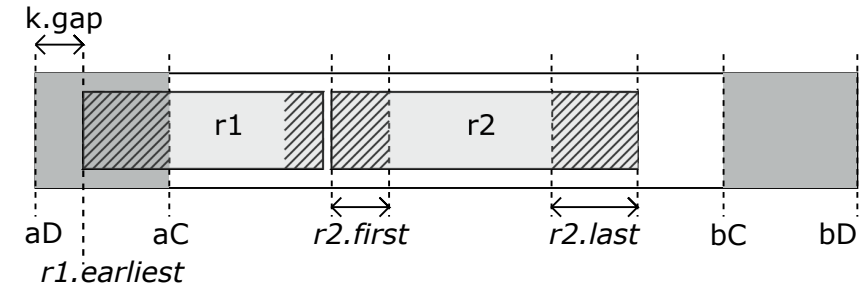
procedure LONGARCLAST(r)
  Arrange route r in a direction such that it ends with larger between $r.first$ and $r.last$

procedure ISFEASIBLESCHEDULE(k, a_D, b_D, a_C, b_C)
  Check if the schedule defined by routes currently assigned to vehicle k, in the given order, violate any time windows

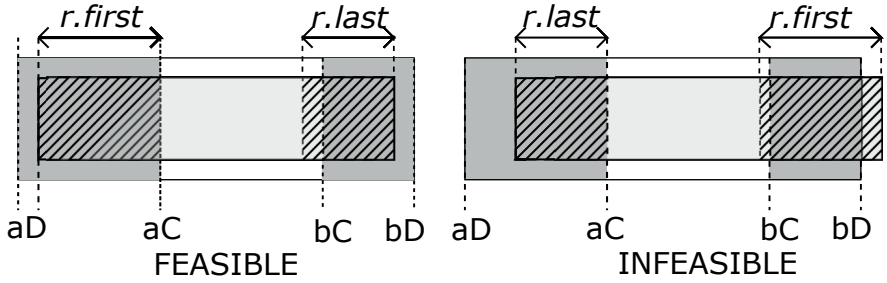
A detailed description of this heuristic is presented in Algorithm 4.1. Lastly, we define the key parameters of this packing heuristic below, which are used in Algorithm 4.1. These definitions are accompanied by a graphical description in Figure 4.7a.

Parameters:

a_D :	opening time of depot.
b_D :	closing time of depot.
a_C :	earliest customer visit allowed.
b_C :	latest customer visit allowed.
$r.first$:	travel time between depot and first customer of trip r.
$r.last$:	travel time between last customer and depot of trip r.
$r.earliest$:	earliest start time for r, $\max\{a_D, a_C - r.first\}$.
$k.gap$:	time gap between a_D and $r.earliest$ of the first trip for k.



(a) Schedule with two trips displaying problem parameters



(b) Infeasible inverting

Figure 4.7: Illustration of route inverting

4.8.3 Comparison of routing methods

Table 4.4 shows the results obtained using i) the branch-and-cut algorithm with an initial heuristic solution and our subtour elimination constraints as the only cuts (4.32)-(4.34), ii) the CPLEX default settings, which automatically decide between dynamic search and branch-and-cut with default CPLEX MIP cuts, depending on the model, and iii) our heuristic method. The columns show the number of customers ($|N|$), the routing cost (*Cost*), the optimality gap between the best upper and lower bounds obtained with the exact procedure (*Gap*), the number of vehicles required ((V), and the run-time (T). The number of cuts found by our separation procedure is also shown for every instance (*Cuts*). In both exact methods, the time limit was set to 1,200 seconds and the optimality gap tolerance was set to 0.001. The results

in bold are the lowest routing costs of the two exact methods. Other variants of our branch-and-cut algorithm were also explored, namely, a combination of subtour elimination cuts and CPLEX cuts (with and without an initial heuristic solution) and subtour elimination constraints without warm start. However, we have not included these results in Table 4.4, as these algorithms were outperformed by our final exact method.

Our heuristic method obtained optimal routing costs in all instances with less than 15 customers, and it often outperformed both exact methods with a 20-minute time limit for larger instances. The number of vehicles obtained is different in some cases, but there does not seem to be a method that consistently requires more vehicles.

The comparison between our branch-and-cut algorithm and CPLEX with default settings looks promising. Even though this comparison gives CPLEX an advantage, our branch-and-cut algorithm outperformed CPLEX in many instances, and, when CPLEX's results were better, this difference was relatively small. Furthermore, we ran several experiments comparing our branch-and-cut algorithm to a branch-and-bound with no cuts. We observed that ours performed significantly better, highlighting the efficiency of our adapted subtour elimination cuts.

Another observation is that our algorithm seems to reduce the peak memory used. Based on several experiments run in AIMMS for instances of up to 25 customers, we observed that after 20 minutes the peak memory was over 60% lower with our subtour elimination cuts with an initial heuristic solution compared to CPLEX's default.

Lastly, Table 4.3 presents a separate computational experiment run on larger instances with 25 to 150 customers (the highest number of customers in a single day), solved using the routing heuristic. The average run time is 30 seconds and all instances were solved in under two minutes.

Number of customers	26-50	51-75	76-100	101-125	126-150
Average run time (s)	2	7	20	38	80
Maximum run time (s)	3	9	24	51	111

Table 4.3: Run-times of heuristic routing for instances for 2018

4.9 Results

Solutions have been generated for each day in the provided data sets and every number of depots from 1 to 10. The resulting costs are summarised in Figure 4.8, showing the average daily costs in 2018 and 2019 for 1 to 10 depots, broken down into depot, fleet, and routing costs. The routing problems are solved for all days in the data sets provided using the heuristic method to speed up the solutions. Due to the long computational times that would have been required to run our hybrid routing algorithm on 730 days, these routing problems have been solved using the heuristic method only. They represent a worst case scenario, as using an exact method for small instances can only improve these results.

There is a clear trade-off between depot costs and fleet and routing costs. As expected, fleet costs contribute the least to the total daily cost due to their low

	Subtour elimination cuts with warm start					Default CPLEX settings				Heuristic		
|N|	Cost ($)	Gap (%)	Cuts (#)	V (#)	T (s)	Cost ($)	Gap (%)	V (#)	T (s)	Cost ($)	V (#)	T (s)
6	**153.1**	≈ 0	6	1	0	**153.1**	≈ 0	1	0	153.1	1	<1s
8	**148.0**	≈ 0	50	1	2	**148.0**	≈ 0	1	0	148.0	1	<1s
9	**185.5**	≈ 0	10	1	7	**185.5**	≈ 0	1	1	185.5	1	<1s
9	**154.0**	≈ 0	115	1	4	**154.0**	≈ 0	1	1	154.0	1	<1s
10	**190.8**	≈ 0	46	1	8	**190.8**	≈ 0	1	2	190.8	1	<1s
11	**136.2**	≈ 0	1,894	1	266	**136.2**	≈ 0	1	240	136.2	1	<1s
12	**216.8**	≈ 0	418	1	39	**216.8**	≈ 0	1	32	216.8	1	<1s
15	**306.4**	13.4	2,448	2	1,200	**306.4**	11.9	2	1,200	306.4	2	<1s
15	**246.2**	22.6	25,919	1	1,200	**246.2**	12.1	1	1,200	253.6	2	<1s
15	**309.1**	30.5	13,821	2	1,200	**309.1**	28.0	2	1,200	309.7	2	<1s
17	**275.0**	34.6	12,499	2	1,200	**275.0**	34.2	2	1,200	311.6	2	<1s
18	**386.5**	22.1	36,389	2	1,200	391.4	21.2	2	1,200	386.5	2	<1s
18	303.8	32.5	13,618	2	1,200	**298.4**	26.3	1	1,200	300.7	2	<1s
19	**244.0**	28.9	11,896	2	1,200	246.6	19.8	2	1,200	251.9	1	<1s
19	**391.5**	75.5	25,862	1	1,200	395.0	22.4	1	1,200	391.5	2	<1s
21	**148.1**	33.2	26,818	1	1,200	**148.1**	30.2	1	1,200	148.1	1	<1s
21	**222.7**	25.6	21,431	1	1,200	223.8	24.3	2	1,200	232.0	1	<1s
22	1,511.1	34.4	16,477	2	1,200	**1,495.1**	75.8	2	1,200	474.4	2	<1s
22	**374.8**	35.3	8,641	2	1,200	414.5	27.1	2	1,200	372.9	2	<1s

Table 4.4: Computational results for January 2018 between 6 and 22 customers

price, and they remain almost constant from five depots onward. Routing costs are the only cost component that varies on a daily basis and, as such, average routing costs are different in 2018 and in 2019. The routing costs in 2019 are slightly lower, although the difference is not significant. Two conclusions can be drawn from this: 2019 demand did not change substantially with respect to 2018, and our methods produce solutions that are robust against unseen data.

Lastly, it is clear that a single depot is not cost-effective and the lowest total costs are obtained for two to five depots. This may be explained by the fact that there are two major cities in the State of Pennsylvania: Pittsburgh and Philadelphia. In Figure 4.9 we illustrate the customer node aggregation using k-medoids to generate 500 clusters from the initial 1,457 customers. The size of the markers indicates the number of days with demand in 2018 for each customer (a) or cluster (b). The colour code serves as guidance to identify the cluster to which each customer was assigned. These urban areas concentrate a significant fraction of the demand such that once a depot is placed in each of these city areas, the marginal benefit of additional depots is greatly reduced.

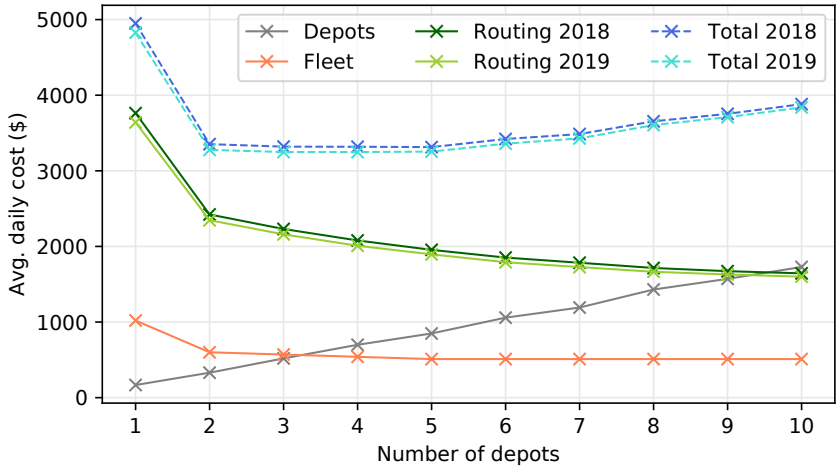


Figure 4.8: Heuristic routing cost breakdown

(a) Original customer locations

(b) Clusters of customer nodes

Figure 4.9: Customer clustering using k-medoids

4.10 Optimizing the number of depots

In this section we develop a framework for automating the choice of the number of depots making use of our existing models and performing an enumeration. To obtain a solution in a reasonable amount of time, we simplified our methodology by i) halving the number of simulated days needed to determine the fleet size ii) designing the fleet for the "P75 demand in 2018" and iii) using for the routing stage the same fast routing approach employed in the fleet sizing model. Furthermore, instead of considering all days in 2018, we selected 15 at random to obtain an average daily routing cost.

Taking into account the trade-off between depot costs and fleet and routing costs, we designed the following iterative scheme. We compute the total costs (depot, fleet and routing) for different numbers of depots, starting from one and increasing the number of depots by one at a time. Every time we obtain a total cost for $|H|$ depots, we compare with the previous cost (for $|H|-1$). If the new total cost is lower, we move on to $|H|+1$ depots; otherwise, we stop: we can conclude that the optimal choice of depots is $|H|$. Using 2018 demand data, this scheme returned a configuration with five depots in just over four minutes of run time.

4.11 Conclusions

We have presented a mathematical model and a solution approach for the multi-trip VRP with time windows, fleet sizing and depot location entered into the 12th AIMMS-MOPTA Optimization Modeling Competition.

Given the complexity of the problem at hand, and assuming a business need to separate strategic and operational decisions, we decomposed this VRP into simpler problems that we solve sequentially. First, we address the location of depots, followed by the fleet sizing (strategic level); then, we determine the routings of a given day (operational level).

The depot location has been treated as a p-median problem. We narrowed down the search by clustering customer nodes using k-medoids as an intermediate step. To determine the fleet size at each depot, we developed a robust methodology that exploited the statistical properties of the demand data to generate feasible solutions for 2018 and 2019. In addition, we give the user the ability to adjust the demand coverage for the level of risk that they are willing to take.

The assignment of customers to depots is decided daily and it takes into account the distance to the depot and the "routing effort" of a customer node from different depots. It was shown that this yields lower routing costs than assigning customers to their nearest depot. For the vehicle routings, we proposed a hybrid approach, namely, we use an exact method for small instances and a fast heuristic for instances with more than 15 customers.

The exact method uses CPLEX and a VRP-specific branch-and-cut algorithm using lifted subtour inequalities, aiming to increase the number of instances on which the exact method can be applied. For the vehicle routing heuristic, we proposed a parametric savings heuristic to encourage routes that are easy to “pack" into vehicles and we developed a packing algorithm that exploits the structure defined by the time windows.

For any number of depots, we were able to determine depot locations, fleet sizes and vehicle routings to serve the demand of each day in 2018 and 2019. We compared several routing methods and the results are promising. Our heuristic method often obtains optimal routing costs in small instances, suggesting that it may provide acceptable results in larger ones. The VRP-specific cuts used outperformed CPLEX in some instances, providing acceptable results in all other instances. These results validate our hybrid approach to the routing problem, and suggest that it is worth exploring the introduction of other VRP-specific cuts to further enhance our method.

Lastly, based on the analysis presented in this paper, we make the following recommendations:

1. *There should be a minimum of two depots.* A single depot would result in a 47% increase in daily costs with respect to two depots due to increased travelling distances.

2. *We recommend to place five depots.* The total costs are very similar for any number of depots between two and five; however, five depots are preferred as this results in the lowest routing costs. This is interesting for two reasons: i) it leads to lower carbon emissions and ii) routing costs are 100% opex, whereas depots have a capex element (e.g. lease or purchase of depots) that can be amortised over time.

3. To reduce fleet costs, it would be worthwhile to *design the fleet size assigned to each depot based on the 75^{th} percentile demand in 2018 and to rent additional vehicles on peak days*. If this is not a viable option, the fleet should be designed using the “max. demand in 2018 + 10%" criterion instead, to ensure complete coverage of demand.

4. The current assumption that no time is required for loading and unloading at the depot nor at a customer location may lead to unplanned driver overtime and customers not receiving their deliveries by 4pm. As an example, in a four-depot scenario with 8.7 customers visited per vehicle in a day and assuming 15 minutes per customer, this would result in two additional hours of driver time per vehicle per day. Two visits to the depot could add one hour on top of that. *We therefore recommend that reasonable values for these parameters be included in the routing model*.

Chapter 5

Conclusion

This thesis focuses on implementations of the Vehicle Routing Problem in complex scenarios. This involves a combination of many distinct aspects that have been individually studied in the literature, from VRP variants to solution methods.

5.1 Content summary and contributions

In Chapter 1 we provide a literature review on the concepts used in the rest of the thesis.

In Chapter 2 we explore a new problem inspired by the operational aspect of a company with a large distribution network. This is a multi-period vehicle routing problem with depot location and route balancing, with trucks and sellers routed simultaneously. We present two MIP models: a 3-index and a 4-index vehicle flow model, and we outline their theoretical advantages. We propose a new form of workload balancing constraints, and new, generalised comb inequalities along with corresponding separation procedures for their identification. Finally, we propose a branch-and-cut algorithm for the 3-index model, tested against the 4-index model solved with the optimization solver CPLEX using a set of 162 instances. The analysis of the results shows that the branch-and-cut algorithm produces better optimality gaps than CPLEX alone in 94% of the tested instances resulting in an average improvement of 64%. Our algorithm is shown to be computationally cheaper to implement, as the number of simplex iterations required is 83% lower compared to the CPLEX method. However, both methods can only handle relatively small instances, hence producing an acceptable feasible solution for instances of over 100 customers would be intractable.

Even though we have managed to reduce the optimality gap obtained by CPLEX with our branch-and-cut procedure, the gap still remains large even for small instances. Our computational experiments show that in 13% of the tested instances the gap is smaller than 5%, and in only 4% of the instances the gap is under 1%. This is largely due to the complexity of the problem, considering there are two routing problems and a location problem performed simultaneously. However, other techniques, out of the scope of this thesis, can potentially improve the quality of the branch-and-cut procedure. Some suggestions are outlined in the next section.

The company in question deals with 1,756 customers. For the purpose of solving

an instance of this size, we developed a heuristic algorithm presented in Chapter 3. We propose an innovative construction heuristic that is based on a cluster-first route-second methodology. The clustering procedure is based on a machine learning clustering algorithm. We propose three variations of this algorithm that can be applied either together or individually to a wide variety of VRP problems. The first variation is a multi-centroid clustering method which results in converging clusters, followed by balanced clustering producing balanced clusters, and finally, joint clustering resulting in intersecting clusters. A random cheapest insertion procedure is applied in each cluster at the routing stage. The construction heuristic alone produces better results than the best-known solution in nearly half of the tested instances in only a fraction of a second. The time difference is incomparable, as the best solution is obtained nearly at the imposed stopping point of two hours on average. A trend is observed that the construction heuristic outperforms the exact solution methods proportionately as the instance size grows.

Our construction heuristic is the starting point of a tabu search implementation, consisting of three improvement stages. The standard course of the tabu search features an inter-route improvement strategy, and the best routes are intensified with a 2-opt procedure. The solution is diversified with a second inter-route strategy, exchanging customers between days. The tabu search procedure results in an average improvement of the solution obtained at the construction stage by 5%, while the time remains under one minute on average. The complete heuristic results in a better solution than the best-known in over 60% of the tested instances.

A solution to the original problem is obtained using an alteration of our proposed heuristic. The construction phase produces a solution within seconds, which during the course of the tabu search algorithm is improved by 1%, while the time increases to 12 minutes.

Our results show that while sophisticated metaheuristics like tabu search can handle large and complex problems in a reasonable time frame, a good quality construction heuristic is invaluable. For large instances of this kind, using only the construction heuristic will produce satisfactory solutions within seconds.

In Chapter 4 we present the findings from a case study involving another complex variant of the Vehicle Routing Problem. We develop a 2-phase solution approach for a multi-trip vehicle routing problem with time windows, fleet sizing and depot location with given demand uncertainties. In the first phase we address the strategic decisions: the location of the depots and the fleet sizes. In the second phase we address the operational decisions involving the routing under some imposed circumstances.

In the first phase, the location of the depots is solved as a p-median problem on a clustered customer set. A robust procedure was developed for fleet sizing. In the second stage, the customers are assigned to vehicles with the account of a predefined routing effort. At this stage, the remainder of the problem consists of a series of standard vehicle routing problems with time windows.

We propose a hybrid solution method for which we developed a mathematical model and a branch-and-cut algorithm to be used for smaller instances, and a specialised heuristic for the larger routes. The branch-and-cut algorithm separates the lifted subtour elimination constraints, which are valid inequalities inserted with a specified strategy. The heuristic approach falls under the category of simultaneous clustering and routing, using a parametric savings heuristic. This procedure is fur-

ther enhanced by a packing heuristic accenting the structure imposed by the time windows. Our proposed heuristic approach often obtains optimal results for the smaller instances, which suggests promising performance for the large instances when no other solution is known. Following the analysis of the results some additional educated recommendations are made for the strategic level of the problem. These include the minimum number and the exact recommended number of depots that should be built, the number of vehicles assigned to each depot to be purchased and a potential number of vehicles to be rented as needed, as well as some additional parameter calibration.

5.2 Further research

From both problems considered in this thesis, it is clear that there is space for improvement in the field of exact solution approaches. Both methods involve a branch-and-cut algorithm which is shown to perform better than an optimization solver alone. However, in both cases, the proposed heuristic approaches often outperform the exact method with an imposed time limit. The run-time difference is incomparable making the heuristic approaches much more desirable for practical use. However, the heuristics themselves can benefit from further research. We propose the following directions:

- Exploration of a wider variety of valid inequalities for the branch-and-cut algorithms. For the problem proposed in Chapter 2, inequalities such as the *mutli-star inequalities* from [5], *framed capacity inequalities* from [100], *hypotour inequalities* implemented in [7] or *closed alternating trail inequalities* used in [42] may be generalised and fitted to the needs of this problem. Their efficiency for the capacitated vehicle routing problem is shown in [91] and more recently for the multi-depot variant in [23].

- The separation procedure proposed for the comb inequalities in Chapter 2 can be improved by exploring other identification strategies. A large variety of separation procedures has already been established in the literature, covering the majority of valid inequalities known. Their successful implementation within the branch-and-cut algorithms proposed for both problems has the potential of significantly improving the performance.

- The second phase of the problem in Chapter 4 can be modelled using a set partitioning formulation (Section 1.5) which can in turn be fitted into a column generation framework (Section 1.6). The Vehicle Routing Problem with Time Windows has been shown to fit well into these solution methods [34].

- The heuristic proposed in Chapter 2 can be further improved. Following the rich literature regarding tabu search algorithms reviewed in [28] and Section 1.7, additional improvement procedures and further parameterisation may result in better quality solutions.

- The second stage of the proposed solution method in Chapter 4 can potentially be tackled with a genetic algorithm. These metaheuristics have been shown to

perform well for similar problems (Section 1.7).

- Stochasticity may be incorporated into the definition of customer demand in Chapter 2. Once a seller visits a customer, they do not have to place an order every time, meaning that the truck may not need to visit all customers as they do under the current definition.

List of Algorithms

List of Figures

List of Tables

Bibliography

[1] Y. Agarwal, K. Mathur, and H. Salkin. “A set-partitioning based exact algorithm for the vehicle routing problem”. In: *Networks* 19 (1989), pp. 731–749.

[2] AIMMS-MOPTA Optimization Modeling Competition. *https://coral.ise.lehigh.edu/~mopta/competition (Accessed November 18, 2020)*. 2020.

[3] D.L. Applegate, R.E. Bixby, and V. Chvátal. *The Traveling Salesman Problem*. Princeton University Press, 2006.

[4] J.R. Araque. *Lots of combs of different sizes for vehicle routing*. Discussion paper. Center for Operations Research and Econometrics, 1990.

[5] J.R. Araque, L.A. Hall, and T.L. Magnanti. *Capacitated trees, capacitated routing and associated polyhedra*. Discussion paper. Center for Operations Research and Econometrics, Catholic University of Louvain, 1990.

[6] J.R. Araque, G. Kudva, T.L. Morin, and J.F. Pekny. “A branch-and-cut algorithm for the vehicle routing problem”. In: *Annals of Operations Research* 50 (1994), pp. 37–59.

[7] P. Augerat. “Approche Polyèdrale du Problème de Tournées de Véhicules”. PhD thesis. Institut National Polytechnique de Grenoble, 1995.

[8] P. Augerat, J.M. Belenguer, E. Benavent, A. Corberán, D. Naddef, and G. Rinaldi. *Computational results with a branch-and-cut code for the capacitated vehicle routing problem*. Research paper. ARTEMIS-IMAG, 1995.

[9] N. Azi, M. Gendreau, and J.-Y. Potvin. “An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles”. In: *European Journal of Operational Research* 202.3 (2010), pp. 756–763.

[10] R. Baldacci, M. Batarra, and D. Vigo. “Valid Inequalities for the Fleet Size and Mix Vehicle Routing Problem with Fixed Costs”. In: *Networks* 54.4 (2009), pp. 178–189.

[11] M. Balinski and R. Quandt. “On an integer program for a delivery problem”. In: *Operations Research* 12 (1964), pp. 300–304.

[12] M.L. Balinski and R.E. Quandt. “On an integer program for a delivery problem”. In: *Operations Research* 12 (1964), pp. 300–304.

[13] J.F. Bard, G. Kontoravdis, and G. Yu. “A Branch-and-Cut Procedure for the Vehicle Routing Problem with Time Windows”. In: *Transportation Science* 36.2 (2002), pp. 250–269.

[14] M. Battarra, M. Monaci, and D. Vigo. “An adaptive guidance approach for the heuristic solution of a minimum multiple trip vehicle routing problem”. In: *Computers and Operations Research* 36.11 (2009), pp. 3041–3050.

[15] J. Baxter. “Depot location: A technique for the avoidance of local optima”. In: *European Journal of Operational Research* 18 (1984), pp. 208–214.

[16] J.E. Beasley. “Route-first cluster-second methods for vehicle routing”. In: *Omega* 11 (1983), pp. 403–408.

[17] T. Bektaş, L. Gouveia, A. Martínez-Sykora, and J.J. Salazar-González. “Balanced vehicle routing: polyhedral analysis and branch-and-cut algorithm”. In: *European Journal of Operational Research* 273 (2019), pp. 452–463.

[18] E. Beltrami and L. Bodin. “Networks and vehicle routing for municipal waste collection”. In: *Networks* 4 (1974), pp. 65–94.

[19] U. Blasum and W. Hochstättler. *Application of the branch-and-cut method to the vehicle routing problem*. Technical report. University of Cologne, 2002.

[20] H.H. Bock. *Selected contributions in data analysis and classification*. Ed. by P. Brito, G. Cucumel, P. Bertrand, and F. de A.T. de Carvalho. Springer Verlag, 2007. Chap. Clustering methods: a history of k-means algorithms, pp. 161–172.

[21] J. Bramel and D. Simchi-Levi. “A location based heuristic for general routing problems”. In: *Operations Research* 43 (1995), pp. 649–660.

[22] J.C.S. Brandão and A. Mercer. “The Multi Trip Vehicle Routing Problem”. In: *Journal of the Operational Research Society* 49 (1998), pp. 799–805.

[23] M.A.J. uit het Broek, A.H. Schrotenboer, B.J. Kees Jan Roodbergen, and L.C. Coelho. “Asymmetric multidepot vehicle routing problems: valid inequalities and a branch-and-cut algorithm”. In: *Operations Research* 69 (2021), pp. 380–409.

[24] N. Christofides, A. Mingozzi, and P. Toth. “Combinatorial Opti-mization”. In: ed. by N. Christofides, A. Mingozzi, P. Toth, and C. Sandi. Wiley, 1979. Chap. The vehicle routing problem, pp. 315–338.

[25] N. Christofides, A. Mingozzi, and P. Toth. “Combinatorial Optimization”. In: ed. by N. Christofides, A. Mingozzi, P. Toth, and C. Sandi. Wiley, 1979. Chap. The vehicle routing problem, pp. 315–338.

[26] V. Chvátal. “Edmonds polytopes and weekly Hamiltonian graphs”. In: *Mathematical Programming* 5 (1973), pp. 29–40.

[27] G. Clarke and J.W. Wright. “Scheduling of vehicles from a central depot to a number of delivery points”. In: *Operations Research* 12.4 (1964), pp. 568–581.

[28] J.F. Cordeau and G. Laporte. “Operations research/computer science interfaces series”. In: ed. by R. Sharda, S. Voß, C. Rego, and B. Alidaee. Vol. 30. Springer, 2005. Chap. Tabu search heuristics for the vehicle routing problem, pp. 145–163.

[29] J.F. Cordeau, G. Laporte, M. Savelsbergh, and D. Vigo. “Transportation”. In: ed. by C. Barnhart and G. Laporte. Vol. 14. Handbooks in Operations Research and Management Science. North-Holland, 2007. Chap. Vehicle Routing, pp. 367–428.

[30] P. Fermín Cueto, I. Gjeroska, A. Solà Vilalta, and M.F. Anjos. “A solution approach for multi-trip vehicle routing problems with time windows, fleet sizing and depot location”. In: *Networks* 78.4 (2021), pp. 503–522.

[31] G. Dantzig, R. Fulkerson, and S. Johnson. “Solution of a large-scale traveling salesman problem”. In: *Journal of the Operations Research Society of America* 2 (1954), pp. 393–410.

[32] G.B. Dantzig and J.H. Ramser. “The Truck Dispatching Problem”. In: *Management Science* 6.1 (1959).

[33] M.S. Daskin and K.L. Maass. “The p-Median Problem”. In: *Location Science*. Ed. by Gilbert Laporte, Stefan Nickel, and Francisco Saldanha da Gama. Basel: Springer International Publishing, 2015. Chap. 2, pp. 21–45.

[34] G. Desaulniers, J. Desrosiers, and M.M. Solomon. *Column generation*. Springer, 2005.

[35] M. Desrochers and G. Laporte. “Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints”. In: *Operations Research Letters* 10 (1991), pp. 27–36.

[36] M. Desrochers and G. Laporte. “Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints”. In: *Operations research letters* 10.1 (1991), pp. 27–36.

[37] M. Desrochers and T.W. Verhoog. *A matching based savings algorithm for the vehiclerouting problem*. Technical Report Cahiers du GERAD G–89–04. École des HautesÉtudes Commerciales de Montréal, 1989.

[38] E.W. Dijkstra. “A Note on Two Problems in Connexion with Graphs”. In: *Numerische Mathematik* 1.1 (1959), pp. 269–271.

[39] E.D. Dolan and J.J. Moré. “Benchmarking optimization software with performance profiles”. In: *Mathematical Programming* 91.A (2002), pp. 201–213.

[40] György Dósa. “The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is FFD(I) ≤ 11/9 OPT(I) + 6/9”. In: *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*. Ed. by Bo Chen, Mike Paterson, and Guochuan Zhang. Springer-Verlag, 2007. Chap. 1, pp. 1–11.

[41] J. Edmonds. “Maximum matching and a polyhedron with 0,1-vertices”. In: *Journal of Research of the National Bureau of Standards* 16B (1965), pp. 125–130.

[42] M. Fischetti. “Facets of the asymmetric traveling salesman polytope”. In: *Mathematics of Operations Research* 16 (1991), pp. 42–56.

[43] M. Fischetti, J.J. Salazar-Gonzáles, and P. Toth. “Experiments with a multi-commodity formulation for a symmetric capacitated vehicle routing problems”. In: *Proceedings of the 3rd meeting of the EURO working group in transportation* (1995).

[44] M. Fischetti and P. Toth. "A polyhedral approach to the asymmetric travelling salesman problem". In: *Management Science* 43 (1997), pp. 1520–1536.

[45] M.L. Fisher and R. Jaikumar. "A generalized assignment heuristic for the vehicle routing problem". In: *Networks* 11 (1981), pp. 109–124.

[46] T.J. Gaskell. "Bases for vehicle fleet scheduling". In: *Operational Research Quarterly* 18 (1967), pp. 281–295.

[47] M. Gendreau, A. Hertz, and G. Laporte. "A tabu search heuristic for the vehicle routing problem". In: *Management Science* 40 (1994), pp. 1276–1290.

[48] M. Gendreau, A. Hertz, and G. Laporte. "New insertion and postoptimization procedures for the traveling salesman problem". In: *Operations Research* 40.6 (1992), pp. 1086–1094.

[49] M. Gendreau, G. Laporte, and J.-Y. Potvin. "The vehicle routing problem". In: ed. by P. Toth and D. Vigo. SIAM, 2002. Chap. Metaheuristics for the capacitated VRP, pp. 129–154.

[50] M. Gendreau, G. Laporte, and R. Séguin. "Stochastic vehicle routing". In: *European Journal of Operational Research* 88 (1996), pp. 3–12.

[51] M. Gendreau and J.-Y. Potvin. *Handbook of metaheuristics*. Springer, 2010.

[52] B.E. Gillett and L.R. Miller. "A heuristic algorithm for the vehicle dispatch problem". In: *Operations Research* 22 (1974), pp. 340–349.

[53] F. Glover. "Future paths for integer programming and links to artificial intelligence". In: *Computers and Operations Research* 13 (1986), pp. 533–549.

[54] B. Golden, A. Assad, L. Levy, and F. Gheysens. "The fleet size and mix vehicle routing". In: *Computers and Operations Research* 11 (1984), pp. 49–66.

[55] B. Golden, A. Assad, L. Levy, and F. Gheysens. "The Fleet Size and Mix Vehicle Routing Problem". In: *Computers and Operations Research* 11.1 (1984), pp. 46–66.

[56] B. Golden, S. Raghaven, and E. Wasil. *The Vehicle Routing Problems, Latest Advances and New Challenges*. Springer, 2008.

[57] B.L. Golden, T.L. Magnanti, and H.Q. Nguyen. "Implementing vehicle routing algorithms". In: *Networks* 7 (1977), pp. 113–148.

[58] L. Gouveia, J. Riera-Ledesma, and J.J. Salazar-Gonzáles. "Reverse multistar inequalities and vehicle routing problems with a lower bound on the number of customers per route". In: *Networks* 61.4 (2013), pp. 281–355.

[59] L. Gouveia, J. Riera-Ledesma, and J. Salazar-González. "Reverse multistar inequalities and vehicle routing problems with lower bound on the number of customers per route". In: *Networks* 61 (2013), pp. 309–321.

[60] L. Gouveia and J. Salazar-González. "Polynomial-time separation of enhanced reverse multistar inequalities". In: *Operations Research Letters* 41 (2013), pp. 294–297.

[61] M. Grötschel and O. Holland. "A cutting plane algorithm for minimum perfect 2-matchings". In: *Computing* 39 (1987), pp. 237–344.

[62] M. Grötschel and M.W. Padberg. “On the symmetric travelling salesman problem I: Inequalities”. In: *Mathematical Programming* 16 (1979), pp. 265–280.

[63] M. Grötschel and M.W. Padberg. “The traveling salesman problem”. In: ed. by E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Wiley, 1985. Chap. Polyhedral theory, pp. 251–306.

[64] E. Hadjiconstantinou, N. Christofides, and A. Mingozzi. “A new exact algorithm for the vehicle routing problem, based on q-paths and k-shortest paths relaxations”. In: *Annals of Operations Research* 61 (1995), pp. 21–43.

[65] S. L. Hakimi. “Optimum locations of switching centers and the absolute centers and medians of a graph”. In: *Operations Research* 12.3 (1964), pp. 450–459.

[66] H.H. Holland. *Adaptation in Natural and Artificial systems. An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. The University of Michigan Press, 1975.

[67] S. Irnich, P. Toth, and D. Vigo. “Vehicle routing: problems, methods and applications”. In: ed. by P. Toth and D. Vigo. 2nd ed. SIAM, 2014. Chap. The family of vehicle routing problems, pp. 1–33.

[68] A.I. Jarrah and J.F. Bard. “Pickup and delivery network segmentation using contiguous geographic clustering”. In: *Journal of the Operational Research Society* 62.10 (2011), pp. 1827–1843.

[69] B. Kallehauge, J. Larsen, O.B.G. Madsen, and M. Marius Solomon. “Vehicle routing problem with time windows”. In: ed. by Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon. Springer, 2005. Chap. Column generation, pp. 67–98.

[70] I. Kara, G. Laporte, and T. Bektaş. “A note on the lifted Miller–Tucker–Zemlin subtour elimination constraints for the capacitated vehicle routing problem”. In: *European Journal of Operational Research* 158 (2004), pp. 793–795.

[71] L. Kaufman and P. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 1990.

[72] G.A.P. Kindervater and M.W.P. Savelsbergh. “Local search in combinatorial optimization”. In: ed. by E.H.L. Aarts and J.K. Lenstra. Wiley, 1997. Chap. Vehicle routing: Handling edge ex-changes, pp. 337–360.

[73] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. “Optimization by simulated annealing”. In: *Science* 220 (1983), pp. 671–680.

[74] K. Knight and J. Hoffer. “Vehicle scheduling with timed and connected calls: A case study”. In: *Operational Research Quarterly* 19 (1968), pp. 299–310.

[75] R.V. Kulkarni and P.R. Bhave. “Integer programming formulations of vehicle routing problems”. In: *European Journal of Operational Research* 20 (1985), pp. 58–67.

[76] J. Kytröjoki, T. Nuortio, O. Bräysy, and M. Gendreau. “An efficient variable neighbourhood search heuristic for very large scale vehicle routing problems”. In: *Computers and Operations Research* 34 (2007), pp. 2743–2757.

[77] G. Laporte. “Fifty years of vehicle routing”. In: *Transportation Science* 43.4 (2009), pp. 408–416.

[78] G. Laporte. “The vehicle routing problem: an overview of exact and approximate algorithms”. In: *European Journal of Operational Research* 59 (1992), pp. 345–358.

[79] G. Laporte. “Vehicle routing: methods and studies”. In: ed. by B.L. Golden and A.A. Asaad. North-Holland, 1988. Chap. Location routing problems, pp. 163–198.

[80] G. Laporte, H. Mercure, and M. Desrochers. “Optimal routing under capacity and distance restrictions”. In: *Operations Research* 133 (1985), pp. 1050–1073.

[81] G. Laporte, H. Mercure, and Y. Nobert. “An exact algorithm for the asymmetrical capacitated vehicle routing problem”. In: *Networks* 16 (1986), pp. 33–46.

[82] G. Laporte and Y. Nobert. “Comb inequalities for the vehicle routing problem”. In: *Methods of Operations Research* 51 (1984), pp. 271–276.

[83] G. Laporte and Y. Nobert. “Exact algorithms for the vehicle routing problem”. In: *Annals of Discrete Mathematics* 31 (1987), pp. 147–184.

[84] G. Laporte, S. Ropke, and T. Vidal. “Vehicle routing problems, methods and applications”. In: ed. by P. Toth and D. Vigo. SIAM, 2014. Chap. Heuristics for the capacitated VRP, pp. 87–116.

[85] G. Laporte and F. Semet. “The vehicle routing problem”. In: ed. by P. Toth and D. Vigo. SIAM, 2002. Chap. Classical heuristics for the capacitated VRP, pp. 109–128.

[86] E.L. Lawler, Jan Karel Lenstra, A.H.G. Rinooy Kan, and D.B. Shmoys. *The traveling salesman problem : a guided tour of combinatorial optimization*. Chichester: John Wiley, 1985.

[87] A.N. Letchford, J. Lysgaard, and R.W. Eglese. “A branch-and-cut algorithm for the capacitated open vehicle routing problem”. In: *The Journal of the Operational Research Society* 58 (2007), pp. 1642–1651.

[88] S. Lin. “Computer solutions of the travelling salesman problem”. In: *Bell System Technical Journal* 44 (1965), pp. 2245–2269.

[89] S. Lin and B.W. Kernighan. “An effective heuristic algorithm for the traveling salesman problem”. In: *Operations Research* 21.2 (1973), pp. 498–516.

[90] J. Lysgaard, A.N. Letchford, and R.W. Eglese. “A new branch-and-cut algorithm for the capacitated vehicle routing problem”. In: *Mathematical Programming* 100 (2004), pp. 423–445.

[91] J. Lysgaard, A.N. Letchford, and R.W. Eglese. “A new branch-and-cut algorithm for the capacitated vehicle routing problem”. In: *Mathematical Programming* 100 (2004), pp. 423–445.

[92] O.B.G. Madsen. “Optimization applied to transportation systems”. In: ed. by H. Strobel, R. Genser, and M. Etschmaier. International Institute for Applied System Analysis, 1976. Chap. Optimal scheduling for trucks – A routing problem with tight due times for delivery, pp. 126–136.

[93] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. Wiley, 1990.

[94] J. McQueen. “Some methods for classification and analysis of multivariate observations”. In: *Computer and Chemistry* 4 (1967), pp. 257–272.

[95] D.L. Miller, A.W. Tucker, and R.A. Zemlin. “Integer programming formulations for traveling salesman problems”. In: *Journal of the ACM* (1960), pp. 326–329.

[96] R.H. Mole and S.R. Jameson. “A sequential route-building algorithm employing a generalized savings criterion”. In: *Operational Research Quarterly* 27 (1976), pp. 503–511.

[97] J.R. Montoya-Torres, S.N. Isaza J.L. Franco, H.F. Jimenez, and N. Herazo-Padilla. “A Literature Review on the Vehicle Routing Problem with Multiple Depots”. In: *Computers Industrial Engineering* 79 (2015), pp. 115–129.

[98] D. Naddef and Y. Pochet. “The traveling salesman polytope revisited”. In: *Mathematics of Operations Research* 26.4 (2001), pp. 700–722.

[99] D. Naddef and G. Rinaldi. “The Vehicle Routing Problem”. In: ed. by P. Toth and D. Vigo. SIAM, 2002. Chap. Branch-and-cut algorithms for the capacitated VRP, pp. 53–84.

[100] D. Naddef and G. Rinaldi. “The Vehicle Routing Problem”. In: ed. by P. Toth and D. Vigo. SIAM, 2002. Chap. Branch-and-cut algorithms for the capacitated VRP.

[101] D. Naddef and S. Thienel. “Efficient separation routines for the symmetric traveling salesman problem I: general tools and comb separation”. In: *Mathematical Programming* 92 (2002), pp. 237–255.

[102] H. Nagamochi, T. Ono, and T. Ibaraki. “Implementing an efficient minimum capacity cut algorithm”. In: *Mathematical Programming* 67 (1994), pp. 325–341.

[103] G. Nagy and S. Salhi. “Location-routing: Issues, models and methods”. In: *European Journal of Operational Research* 177 (2007), pp. 649–672.

[104] M.D. Nelson, K.E. Nygard, J.H. Griffin, and W.E. Shreve. “Implementation techniques for the vehicle routing problem”. In: *Computers and Operations Research* 12 (1985), pp. 273–283.

[105] Andrei Novikov. “PyClustering: Data Mining Library”. In: *Journal of Open Source Software* 4.36 (2019), p. 1230.

[106] M. Padberg and S. Hong. “On the symmetric travelling salesman problem: a computational study”. In: *Mathematical Programming Study* 12 (1980), pp. 78–107.

[107] M. Padberg and G. Rinaldi. “Facet identification for the symmetric travelling salesman polytope”. In: *Mathematical Programming* 47 (1990), pp. 219–257.

[108] M. Padberg and T.-Y. Sung. "An analytical comparison of different formulations of the traveling salesman problem". In: *Mathematical Programming* 52 (1991), pp. 315–357.

[109] M.W. Padberg and M.R. Rao. "Odd Minimum Cut-Sets and b-Matchings". In: *Mathematics of Operations Research* 7.1 (1982), pp. 67–80.

[110] H. Paessens. "The savings algorithm for the vehicle routing problem". In: *European Journal of Operational Research* 34 (1988), pp. 336–344.

[111] C.H. Papadimitriu and K. Steiglitz. *Combinatorial optimization*. Dover Publications Inc., 1998.

[112] K. Paton. "An algorithm for blocks and cutnodes of a graph". In: *Communications of the ACM* 14.7 (1971).

[113] A. Pessoa, M. Poggi de Aragão, and E. Uchoa. "A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem". In: *Networks* 54.4 (2009), pp. 178–189.

[114] C. Prins. "A simple and effective evolutionary algorithm for the vehicle routing problem". In: *Computers and Operations Research* 31 (2004), pp. 1985–2002.

[115] H. Pullen and M. Webb. "A computer application to a transport scheduling problem". In: *Computer Journal* 10 (1967), pp. 10–13.

[116] T.K. Ralphs, L. Kopman, W.R. Pulleyblank, and L.E. Trotter. "On the capacitated vehicle routing problem". In: *Mathematical Programming* 94 (2003), pp. 343–359.

[117] E. Russell and W. Igo. "An assignment routing problem". In: *Networks* 9 (1979), pp. 1–17.

[118] F. Semet, P. Toth, and D. Vigo. "The Vehicle Routing Problem". In: ed. by P. Toth and D. Vigo. second. SIAM, 2014. Chap. Algorithms for the capacitated vehicle routing problem, pp. 37–57.

[119] E. Taillard. "Parallel iterative search methods for vehicle routing problems". In: *Networks* 23 (1993), pp. 661–673.

[120] P.M. Thompson and H.N. Psaraftis. "Cyclic transfer algorithms for multivehicle routing and scheduling problems". In: *Operations Research* 41 (1993), pp. 935–946.

[121] F.A. Tillman. "The multiple terminal delivery problem with probabilistic demands". In: *Transportation Science* 3 (1969), pp. 192–204.

[122] P. Toth and D. Vigo. "Models, relaxations and exact approaches for the capacitated vehicle routing problem". In: *Discrete Applied Mathematics* 123 (2002), pp. 487–512.

[123] P. Toth and D. Vigo. "The Vehicle Routing Problem". In: ed. by P. Toth and D. Vigo. SIAM, 2002. Chap. Branch-and-bound algorithms for the capacitated VRP, pp. 29–51.

[124] Trading Economics. *https://tradingeconomics.com/united-kingdom/gdp-from-transport (Accessed October 6, 2022)*. 2020.

[125] D. Tuzun and L.I. Burke. “A two-phase tabu search approach to the location routing problem”. In: *European Journal of Operational Research* 116 (1999), pp. 87–99.

[126] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian. “New benchmark instances for the capacitated vehicle routing problem”. In: *European Journal of Operational Research* 257 (2017), pp. 845–858.

[127] P. Wark and J. Holt. “A repeated matching heuristic for the vehicle routing problem”. In: *Journal of Operational Research Society* 45 (1994), pp. 1156–1167.

[128] A. Wren. *Computers in Transport Planning and Operation*. Ian Allan, 1971.

[129] P. Yellow. “A computational modification to the savings method of vehicle scheduling”. In: *Operational Research Quarterly* 21 (1970), pp. 281–283.

[130] E.E. Zachariadis and C.T. Kiranoudis. “A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem”. In: *Computers and Operations Research* 37 (2010), pp. 2089–2105.

Appendices

Appendix A

VRP formulations

There are three main types of formulations for the VRP that can be found in the literature, defined on a graph $G(N, E)$, where N is the set of nodes and E is the set of edges:

- **Vehicle Flow Formulations** use integer variables to represent the number of times a vehicle has traversed an edge. This is an intuitive integer programming formulation, that arises from the basic TSP formulations. When it comes to applying exact methods to the problem, it has been proven to be a useful model often found in the literature associated with branch and cut approaches. The vehicle flow model is especially useful when the objective cost is calculated as a sum of the edge lengths. However, when it comes to variants that are common in practice, even small changes are hard to fit into the model efficiently. For example, when different types of vehicles are on disposal, or when the node sequence is taken into account in the overall cost of the solution. Moreover, the linear relaxation of this problem can be very weak depending on the additional constraints.
- **Commodity Flow Formulations** there are additional variables associated with the edges representing the flow of the commodities passing through.
- **Set Partitioning Formulations**, or often found as a *Set Partitioning Problem (SPP)* is the problem where an exponential number of binary variables is used, each representing a *feasible circuit*. The advantage of this formulation is that it only deals with feasible routes, therefore the side constraints that control the capacity are not taken into account in the main model. The problem objective is to find an optimal combination of feasible routes, such that all of the customer demand is satisfied. The downside of the problem, however, is the large number of variables.

A.1 Vehicle Flow Model

The basic Vehicle Flow Model is a 2-index formulation that uses $O(n^2)$ variables, all binary, which indicate whether or not a vehicle traverses a given edge in the optimal solution.

$$x_{ij} = \begin{cases} 1, & \text{if a vehicle traverses edge } (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$$

The basic model is formulated as follows:

$$\min \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij}$$

$$\text{s.t.} \sum_{i \in N} x_{ij} = 1, \qquad \forall j \in N \backslash \{0\}, \tag{A.1}$$

$$\sum_{j \in N} x_{ij} = 1, \qquad \forall i \in N \backslash \{0\}, \tag{A.2}$$

$$\sum_{i \in N} x_{0j} = K, \tag{A.3}$$

$$\sum_{i \in N} x_{i0} = K, \tag{A.4}$$

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq r(S), \qquad \forall S \subseteq N \backslash \{0\}, S \neq \emptyset, \tag{A.5}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i,j \in N. \tag{A.6}$$

Constraints (A.1) and (A.2) are known as the *indegree* and *outdegree* constraints. They represent the condition that every customer needs to be visited by a vehicle exactly once. Constraints (A.3) and (A.4) imply that the number of vehicles leaving from and returning to the depot has to be exactly K. These are the degree constraints of the depot. Constraint set (A.5), known as *capacity-cut constraints (CCC's)* impose both the capacity and connectivity requirements. $r(S)$ is the minimum number of vehicles needed to satisfy the demand of all of the customers in the set S. It is usually determined by either solving the *bin-packing problem* or by finding its trivial lower bound for simplicity. Constraints (A.6) define the binary variables.

Constraint A.5 is often replaced with the following:

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - r(S) \qquad \forall S \subseteq V \backslash \{0\}, S \neq \emptyset$$

A more common formulation is the single index formulation. In this case the problem is defined in an undirected graph $G(N,E)$, where the edges are denoted as $(i,j) = (j,i) = e \in E$, $i,j \in N$. The decision variables are defined as

$$x_e = \begin{cases} 1, & \text{if a vehicle traverses edge } e \in E \\ 0, & \text{otherwise} \end{cases}$$

The problem is formulated as follows:

$$\min \sum_{e \in E} c_e x_e$$

$$\text{s.t.} x(\delta(i)) = 2, \qquad \forall i \in N\backslash\{0\}, \tag{A.7}$$

$$x(\delta(S)) \geq 2r(S), \qquad \forall S \subseteq N\backslash\{0\}, |S| \geq 2, \tag{A.8}$$

$$x_e \in \{0,1\}, \qquad \forall e \notin \delta(0), \tag{A.9}$$

$$x_e \in \{0,1,2\}, \qquad \forall e \in \delta(0). \tag{A.10}$$

Here, $x(\delta(i)) = \sum_{e\in\delta(i)} x_e$ and, analogously $x(S) = \sum_{e\in S} x_e$. As before, (A.7) represent the *degree constraints* - each customer is visited only once. (A.8) are the *capacity inequalities* representing the restrictions imposed by the capacities of the vehicles. They also ensure that the vehicle routes are connected. Since the BPP is $\mathcal{NP}$-hard, calculating $r(S)$ is generally avoided. In the literature, again, this is substituted by $k(S) = \lceil d(S)/C \rceil$, which is an obvious lower bound. When this is used instead of $r(S)$, (A.8) are referred to as *rounded capacity inequalities* (RCS). Constraints (A.9) and (A.10) are the integrality conditions. Here we are allowing some decision variables to take the value of 2. This means that if a customer i has a high demand $q_i \sim Q$, we are allowing a vehicle to visit that customer only and then return to the depot, which means the edge between customer i and the depot will be traversed twice by the same vehicle.

In this case as well, constraint set (A.8) can be substituted by the alternative generalised subtour elimination constraints.

$$\sum_{e\in E(S)} x_e \leq |S| - r(S), \qquad \forall S \subseteq V\backslash\{0\}, S \neq \emptyset.$$

Some variations of the VRP involving a heterogenous vehicle fleet or similar extensions, require an additional index to specify the vehicle in question. The 3-index formulation makes use one set of $O(n^2K)$ binary variables:

$$x_{ijk} = \begin{cases} 1, & \text{if vehicle } k \text{ traverses edge } (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$$

and another set of $O(nK)$ binary variables

$$y_{ik} = \begin{cases} 1, & \text{if customer } i \text{ is served by vehicle } k \\ 0, & \text{otherwise} \end{cases}$$

The formulation is as follows:

$$\min \sum_{i\in N} \sum_{j\in N} c_{ij} \sum_{k=1}^{K} x_{ijk}$$

$$\text{s.t.} \sum_{k=1}^{K} y_{ik} = 1, \qquad \forall i \in N\backslash\{0\}, \tag{A.11}$$

$$\sum_{k=1}^{K} y_{0k} = K, \tag{A.12}$$

$$\sum_{j\in N} x_{ijk} = \sum_{j\in N} x_{jik} = y_{ik}, \qquad \forall i \in N, k = 1,\dots,K, \tag{A.13}$$

$$\sum_{i\in N} d_i y_{ik} \leq C, \qquad \forall k = 1,...,K, \tag{A.14}$$

$$\sum_{i\in S}\sum_{j\notin S} x_{ijk} \geq y_{hk}, \qquad \forall S \subseteq N\backslash\{0\}, h \in S, k = 1,\dots,K, \tag{A.15}$$

$$x_{ijk} \in \{0,1\}, \qquad \forall i,j \in N, k = 1,\dots,K, \tag{A.16}$$

$$y_{ik} \in \{0,1\}, \qquad \forall i \in N, k = 1,\dots,K. \tag{A.17}$$

Constraints (A.11) and (A.12) are the degree constraints of the customers and the depot respectively. Constraint (A.13) imposes that the same vehicle that serves a given customer, also enters and leaves the customer. Constraint (A.14) controls the capacity in each route, and constraint (A.15) controls the connectivity. The remaining two sets of constraints define the binary variables.

Again, in this formulation the connectivity constraints (A.15) can be replaced by

$$\sum_{i\in S}\sum_{j\in S} x_{ijk} \leq |S| - 1, \qquad \forall S \subseteq V\backslash\{0\}, |S| > 2, k = 1,...,K.$$

A.2 Commodity Flow Model

As the name suggests, in this formulation, other than the number of times an edge is traversed by a vehicle, the commodity flow through the edge is also important. For this purpose a new set of continuous variables is introduced. A description of this problem can be found in [67]. Even though it has been proven to be useful in many practical applications often found in the gas industry, this model is not used in exact approaches in the literature.

Firstly, an extended graph is defined $G' = (N', E')$, where a duplicate of the depot is taken into account: $N' = N \cup \{0\} \cup \{n+1\}$. The decision variables are:

$$x_{ij} = \begin{cases} 1, & \text{if a vehicle traverses edge } (i,j) \in E' \\ 0, & \text{otherwise,} \end{cases}$$

and the set of continuous, non-negative commodity flow variables:

$y_{ij} > 0$, vehicle load capacity when a vehicle traverses edge (i,j)

$y_{ji} = C - y_{ij}$, vehicle residual capacity when a vehicle traverses edge (i,j)

The role of the flow variables is to represent two separate paths for any feasible route: one from node 0 to $n+1$ giving the vehicle load, and one from $n+1$ to 0 giving the residual flow.

The formulation is as follows:

$$\min \sum_{(i,j)\in A'} c_{ij}x_{ij}$$

$$\text{s.t.} \sum_{j\in N'} (y_{ji} - y_{ij}) = 2d_i, \qquad \forall i \in V'\backslash\{0, n+1\}, \tag{A.18}$$

$$\sum_{j\in N'\backslash\{0,n+1\}} y_{0j} = d(N'\backslash\{0, n+1\}), \tag{A.19}$$

$$\sum_{j\in N'\backslash\{0,n+1\}} y_{j0} = KC - d(N'\backslash\{0, n+1\}), \tag{A.20}$$

$$\sum_{j\in N'\backslash\{0,n+1\}} y_{n+1,j} = KC, \tag{A.21}$$

$$y_{ij} + y_{ji} = Cx_{ij}, \qquad \forall (i,j) \in A', \tag{A.22}$$

$$\sum_{j\in N'} (x_{ij} + x_{ji}) = 2, \qquad \forall i \in N'\backslash\{0, n+1\}, \tag{A.23}$$

$$y_{ij} \geq 0, \qquad \forall (i,j) \in A', \tag{A.24}$$

$$x_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A'. \tag{A.25}$$

Constraint (A.18) implies that the difference between the flow entering and leaving a given customer needs to be equal to twice its demand. Constraints (A.19)–(A.21) impose the amount of flow on the edges incident to the depot. Constraint (A.22) defines the flow variables as $y_{ij} = C - y_{ji}$, and constraint (A.23) imposes that every customer node is entered and left exactly once. The remaining two constraints define the decision variables as non-negative and binary respectively.

A.3 Set Partitioning

The set partitioning formulation takes the approach of generating the set of all circuits in $G = (N, A)$, $\mathscr{H} = \{H_1, H_2, ..., H_q\}$, each corresponding to a feasible route. The cost of each circuit H_j is denoted by c_j. The decision variables in this problem are:

$$x_i = \begin{cases} 1, & \text{if a route } i \text{ is in the optimal solution} \\ 0, & \text{otherwise} \end{cases}$$

The problem also makes use binary coefficients a_{ij}, $i \in N\backslash\{0\}$, $j \in \{1, ..., q\}$, defined as

$$a_{ij} = \begin{cases} 1, & \text{if a customer } i \text{ belongs in route } j \\ 0, & \text{otherwise} \end{cases}$$

The formulation of this problem is as follows:

$$\min \sum_{j=1}^{q} c_j x_j$$

$$\text{s.t.} \sum_{j=1}^{q} a_{ij}x_j = 1, \qquad \forall i \in V\backslash\{0\}, \tag{A.26}$$

$$\sum_{j=1}^{q} x_j = K, \tag{A.27}$$

$$x_j \in \{0,1\}, \qquad \forall j = 1,...,q. \tag{A.28}$$

$$\tag{A.29}$$

Constraint (A.26) implies that every customer needs to belong in exactly one circuit. Constraint (A.27) means that there are exactly K circuits in the optimal solution. The last constraint defines the binary variables.

If constraint (A.26) is replaced by

$$\sum_{j=1}^{q} a_{ij} x_j \geq 1, \qquad \forall i \in N \backslash \{0\},$$

the problem is transformed into a *set covering* problem. Every feasible solution of the first problem is also feasible for the second. Furthermore, a feasible solution for the second problem may easily be transformed into a feasible solution for the first problem. If a solution is infeasible, that means that at least one customer is visited more than once. This can be then easily fixed by removing this customer for every route from the optimal solution, except for one. If the triangle inequality holds, this change will only reduce the cost of the route.

Using the second formulation is more practical than using the first, as this way among all of the feasible routes, only those with the largest number of customers included need to be considered. This way the number of variables is reduced, and the dual formulation only considers non-negative variables.

This model is interesting because of its compact formulation. The one downside is the huge number of binary variables used - even with instances with a few tens of customers, the number of variables can go up to a billion. The generation of feasible routes is difficult and time consuming, and the linear relaxations need to be solved using column generation approaches.

Appendix B

Identifying Blocks

An algorithm for block identification in an arbitrary undirected graph was introduced by Paton in the 70s [112]. This polynomial time algorithm grows a spanning tree and systematically labels or re-labels edges. Two different edges at the end of the algorithm bear different labels if and only if they belong to different blocks.

Let $G = (N, E)$ be a connected graph where $N = \{1, 2, ..., n\}$ is the set of nodes. Let T be a tree in G with the arbitrary node $r \in N$ as the root. At any stage of the tree, the set of nodes will be partitioned as $N = R \cup X \cup U$ where R represents the nodes not yet in the tree T, X is the set of explored nodes in the tree and U is the set of nodes from the tree not yet explored. For every $i \in N \backslash \{r\}$ we define a predecessor $p(i)$ – a unique node such that $(p(i), i) \in T$ and $p(i)$ is closer to r than i is. We define a distance $d(i)$ for every node $i \in X$ as $d(i) = d(p(i)) + 1$. The idea of the algorithm is to assign labels to the edges of T. We denote these by $b(i)$ for edge $(p(i), i) \in T$. When the algorithm terminates, if nodes $i, j \in N$ belong to the same block, then $b(i) = b(j)$. The labels are updated during the process of the algorithm. In order to keep track of changes, we define a Boolean n-tuple $A \in [0, 1]^n$.

Initialise Choose a node $r \in R$ to be the root of T. Set $R = N \backslash \{r\}$, $X = \emptyset$ and $U = \{r\}$. Set $b(i) = 0$ and $A(i) = 0$ for all $i \in R$. Set cycle size $L = 0$.

Step 1 If $U = \emptyset$ go to Step 5. Otherwise, take $v = \min_i \{i \in U\}$ and set $U = U \backslash \{v\}$, $X = X \cup \{v\}$. If $L > 0$, set $L = L - 2$.

Step 2 For every edge $(v, z) \in E$ such that $z \in U$, there is a cycle made up of (v, z), $(z, p(z))$ and the existing path from v to $p(z)$ in T. Change $b(z)$ from its former value q to v. If $q > 0$, set $A(q) = 1$ to note this change of label. Set $L = \max\{L, d(v) - d(p(z))\}$.

Step 3 For every edge $(v, z) \in E$ such that $z \in R$, add (v, z) to the tree T and set $R = R \backslash \{z\}$, $U = U \cup \{z\}$, $p(z) = v$, $d(z) = d(v) + 1$ and set a negative label $b(z) = -z$.

Step 4 If $L = 0$ no cycles have been found – go to Step 5. Otherwise, for all edges $(p(k), k)$ in the cycle of length L in T from v to r, set $A(b(k)) = 1$. For every edge $(p(k), k) \in T$ for which $A(b(k)) = 1$, set the label $b(k) = v$ to indicate that they belong to the same block. Set $X = X \cup \{v\}$, $U = U \backslash \{v\}$ and go to Step 1

Step 5 Renumber all labels $b(i)$ in a consecutive order $1, 2, ..., s$. Terminate.

At the end of the algorithm, if in Step 5 $s = 1$, the graph G itself is a block. Otherwise, cutnodes exist and these can be identified as nodes that appear on edges in T with different labels. All edges carrying the same label $b(i)$ belong to the same block i.