Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

University of Padua

Department of Information Engineering

Master Degree in
Computer Engineering

A Matheuristic Approach to a real life Vehicle Routing Problem

Supervisor:
Professor Roberti Roberto

Student:
Pastore Alessandra
2053082

Company Tutor:
Doctor Bortolomiol Stefano

Academic Year 2023/2024
Graduation Date: 07/03/2024

Abstract

The domain of logistics involves managing the flow of goods to meet the customer needs. The Vehicle Routing Problem (VRP) is a well-known problem in the field of Operation Research, often used to model and efficiently solve logistics problems. Real life VRPs present various combination of constraints to model different scenarios, making them more challenging to manage than classic VRPs described in the literature. This paper proposes a matheuristic based algorithm to solve large-scale instances of a real life VRP enriched with popular constraints such as Time Windows or the use of a heterogeneous fleet of vehicles. The key step of the proposed algorithm is the optimization of a Set Covering Problem, making it highly adaptable to tackle the complexities of real life VRPs with different constraints. The algorithm was tested with benchmark instances up to 1000 customers and compared to the more competitive ALSN algorithm developed by Optit S.r.l.

Index

Chapter 1

Introduction

The problem of logistics and efficient transportation for distributing goods is one of the main study topics of the field of *Operations Research* (OR). Logistics is the discipline that manages the flow of goods and supply from one facility to another to satisfy some customer needs. Other than the routing process of supplies, key activities of logistics also comprehend restock, future planning, tracking of goods and storage managing. A good logistics system tries to achieve its goal by optimizing delivery times, transportation costs and resources, in order to reduce waste while improving customer service and satisfaction. Logistics is used by many organizations in daily activities such as garbage collection, mail delivery, public transportation and supply restock for businesses.

Examples of multinational corporations that makes an intense use of logistics there is Amazon, Wal-Mart, Aldi Group and Carrefour. In Italy we can find Coop and Conad, together with Poste Italiane, BRT and DHL, as stated by AGCOM [1].

OR is used when there's the need to make complex decisions that can be solved using mathematical methods. A logistics problem is a perfect example of decisions making that need to be optimized minimizing costs and maximizing efficiency.

The challenge of efficient transportation of goods makes arise the so-called *Vehicle Routing Problem* (VRP) that focuses on the optimization of routing design from some origin points to multiple destinations. Its main goal is to determine the most cost-efficient routes to take to provide services or deliver goods to a set of geographically dispersed customers, given a set of depots and a fleet of vehicles, without the violation of problem-specific constraints (Braekers et al. [5]) - e.g. travel distances, time limits, time windows, vehicles capacity and more factors.

It's important to introduce some terms that refer to the VRP and that will be recurrent during the entire paper.

In the case of VRP we call a *depot* the origin point from where the flow of goods starts and *customer* (or *client*) a point of delivery. A *Transport Request* can also be used when talking about a point of delivery, meaning that a customer must be served. A *route* is sequence of customers that are visited consecutively by a *vehicle* starting and ending at a depot. Referring to the convention of Cattaruzza et al. [8], different terms can be used by different authors to talk about the same structures, such as *trip* and *journey*.

Figure 1.1 shows an example of a simple VRP instance, where some customers are served by three different vehicles (the red, green and blue route) departing from the same depot. Each route serves four different customers and the total cost of a route is given by the sum of the arcs that connect each node of the graph.

On recent decades, this problem has been researched and studied to improve its possible application over real life problems, thus incorporating real life constraints. Toth and Vigo [43] cover in their work most of the VRP variants and their practical issues depending on such parameters and constraints. It is used by many companies to solve the problem of road transportation targeting costs, fuel consumption, customer satisfaction and other important aspects for the managing of a company. For those companies, solving the transportation problem means improving the work-life balance of their employees, reducing the driving time, fuel costs and gaining more profits.

The VRP was firstly introduced in 1959 by Dantzig and Ramser [13] as the "Truck Dispatching Problem" to solve the problem of transportation of oil from a central hub (the depot) to some gas stations (the customers). It was firstly addressed as a generalization of the *Traveling Salesman Problem* (TSP) where the salesman was required to return to the "terminal point" (the depot) every m point visited. Introducing the condition of capacity of the salesman, the problem becomes really similar to the VRP we know today.

In 1964 it was modeled by Clarke and Wright [11] and then solved using a greedy approach. After that, many new approaches were proposed to find both optimal and approximate solution to the problem, such as Local Search and Simulated Annealing, as well as presenting many new models and variants more relevant to real life routing problems.

It is important to make clear that the VRP is not the only optimization problem used to solve efficient logistics. VRP is often used for planning deliveries and optimizing routing where multiple customers must be satisfied. It can be integrated with other formulations such as the TSP to find intra route optimality. In fact VRP

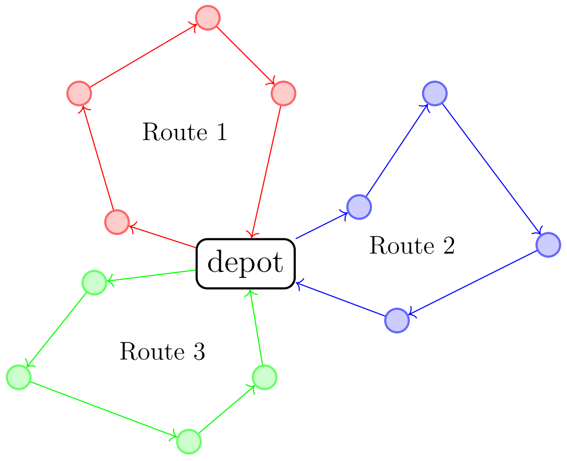


Figure 1.1: Example of VRP with a single depot and three routes that serve different customers.

can be seen as multiple TSP to be solved (one per route) once all customers are assigned to different routes and vehicles, hence the problem becomes how to cluster the customers and only then how to visit them in the single routes. Noteworthy are also the Knapsack Problem and the Bin Packing Problem. Both can be used when there's the need to allocate goods having limited vehicles with a limited capacity and both can optimize supply stocking and storage usage with large demands from customers.

Current used VRP models need to reflect real life challenges, and are significantly different from the standard formulation, with an increasing complexity of constraints that need to mimic traffic congestion, demand information and pickup and delivery times. Classical variants as the *Capacitated Vehicle Routing Problem* (CVRP), or the *VRP with Time Windows* (VRPTW), are classified as specific optimization problems families that are still very popular in the community (Caceres-Cruz et al. [6]) and are still present as constraints in real case studies. In the CVRP the vehicles have a limited capacity and the requests must be assigned based on their quantity (weight, volume...) without violating the capacity of the vehicle. In the VRPTW each request must be served within a certain time windows and must account for service time or idle times. A more detailed description of some VRP variants can be found in Chapter 3.

While the current VRP literature is rather vast and often focuses only on specific variants of the VRP, there are taxonomies as Braekers et al. [5] and overview articles as Toth and Vigo [42, 43], or Golden et al. [22] that could help understand the development of the field in order to better approach the real life problem introduced with this thesis, see Chapter 4.

1.1 Objectives

The VRP is a known NP-hard problem and as such it cannot be optimally solved by an algorithm in polynomial time (see Lenstra and Kan [26]). Exact algorithms can only be used over small instances of the problem, but they become highly inefficient over real life instances, where the size is considerably large. Only heuristics and metaheuristics algorithms are suitable to solve real life VRP, meaning that there's no guarantee of finding an optimal solution.

The work presented with this thesis has been carried out in collaboration with the company Optit S.r.l. [39], which specializes in the development and design of OR-based software. Their expertise lies in optimizing routing, scheduling, and various decision-making processes within logistics and other industrial domains.

The aim of this paper is to use both heuristics and exact methods to develop an algorithm to solve a rich VRP that arises in a real life distribution logistics application commissioned by one of Optit clients. The main focus of the algorithm is to solve the real life VRP using the Set Covering (SC) formulation as the main step, leading to a modular algorithm that can be easily adapted to different variants of the VRP.

One of the main objectives of this research project is the comparison between this SC-based algorithm and the algorithm currently used by the company, which is largely based on an Adaptive Large Neighborhood Search (ALNS) metaheuristic based over Pisinger and Ropke LNS [32].

This ALNS algorithm works by applying different refinement operators at each iteration based on the solution quality and operator efficiency of previous iterations. An example of common operators from the literature can be ruin-and-recreate applied to both single routes or the entire solution, or the operators for swap, replace and k-opt moves applied intra-route and inter-route.

The instances used for the comparison are based on data provided by one of Optit's clients. The real life scenario taken into consideration is the delivery of goods for the HORECA (Hotel, Restaurant and Cafè or Catering) sector, that encompasses the whole food service industry.

The main problems to consider are the distribution of the customers and their demands. Within the HORECA sector, the distribution area to account for comprehends both city center, where only smaller vehicles can enter, and areas that can also be served by larger vehicles. Different municipalities and districts can also dictate which vehicles can operate and which not, adding more incompatibilities and constraints to the problem.

While an ALNS algorithm must consider the feasibility of a solution and must adapt during each iteration to be able to always find better solutions, the SC approach can focus more on the generation of good sets of routes.

1.2 Structure of the Thesis

This paper has the following structure. Section 2 reviews some papers taken from the literature on algorithms using the Set Covering or Set Partitioning formulation to solve the VRP and other heuristic methods to solve large instances. Section 3 formally define the classic VRP with an arc-based formulation, describes its main variants and states the Set Covering Problem formulation applied to the VRP. The VRP variant that is going to be the main study case of this thesis is described in Section 4. Section 5 describes in details the algorithm developed and the choices that were made to implement it efficiently. The final results can be found in Section 6 to show a direct comparison of the performance of this approach versus the ALNS algorithm. The conclusion Section 7 introduces some possible future work and a final analysis.

Chapter 2

Literature Review

Following is a concise overview of the algorithms that solve the VRP using a Set Partitioning or Set Covering formulation that hold particular relevance to this thesis. A small section is dedicated to some papers on decomposition strategies, constructive algorithms and heuristics that were used in this thesis algorithm.

Since our study case is a real life VRP that includes multiple constraints, it is difficult, if not nearly impossible, to find the exact same VRP variant in the literature. Most proposed algorithms in the literature focus on specific variants of the VRP. While relevant for academic study cases, this approach is very unpractical in real life scenarios. To solve a rich VRP, one of the best strategy to maintain a highly flexible algorithm that tackles different constraint combinations, is to develop hybrid algorithms that solve a Set Partitioning or Set Covering model. The main idea is to delegate the generation of feasible routes to an initial phase of the algorithm while letting routes combination and the solution optimization to the mathematical model.

2.1 SP and SC based Algorithms

Balinski and Quandt [3] introduce the idea of the so-called Petal Algorithms, consisting in the solution of a Set Partitioning problem with routes as input objects, in order to find a solution to the VRP. Being impractical due to the large number of columns in the mathematical problem, Petal Algorithm revolved to solving the SP heuristically.

Many approaches can be applied to solve the SP problem, with *Column Generation* as one of the most popular strategies. A good pricing technique and the use of heuristics led Caprara et al. [7] to winning the FASTER competition, organized in 1994 by Ferrovie dello Stato SpA and the e Italian Operational Research Society, which aimed at crew scheduling for railway applications.

As already stated, the use of Column Generation to overcome the computational complexity of the *Integer Linear Programming* (ILP) formulation of the SP is one of the most used techniques in the literature. In 1976, Foster and Ryan [19] described this approach applied to a VRP with constraints inspired by real world vehicle scheduling problem, with major interest over multiple days planning.

Another popular approach is to use Local Search heuristics together with the SP formulation. Rochat and Taillard [34] carry out this technique to solve efficiently the VRP by focusing on Tabu Search. The article demonstrates how an extensive use of this well-known heuristic, together with the solution of a SP problem as a post optimization technique, led to the improvement of nearly forty problem instances taken from the literature.

Other works have followed the example of using the SP formulation for dealing with different VRP variants. Kelly et al. [23] study the use of simple constructive and improvement heuristics against the use of more competitive algorithms. The choice of using a simple Local Search makes the code more flexible and faster, more suited to the use of an SP formulation and to solve a rich VRP. The authors actually use a relaxation of the SP that highly reminisce a Set Covering formulation, in which over-coverages of customers may appear. This small difference implies that the solution found by the model is not considered feasible and that the need to develop a Patching heuristic to remove multi-covered customers from all but one route arises.

Not many works in the literature use the SC formulation instead of the SP one. The main disadvantage is the need to add a subroutine that removes the extra customers from all the routes but one, to regain feasibility in the solution. It's more common to find paper that address the SP methodology while also allowing the possible use of an SC formulation instead.

De Franceschi et al. [20] model a new heuristic extending a procedure proposed for the TSP in 1981 by Sarvanov and Doroshko. The aim of the paper is to address the Distance-Constrained Capacitated VRP through the use of an underling ILP that solve an SP formulation. However, it is also stated that the same procedure can be applied to an ILP having the structure of an SC, resulting in an easier problem to solve that only needs a simple reallocation subroutine for the multi-covered customers.

Overall, the majority of the papers analyzed propose modular algorithms that aggregate iteratively an initial Local Search with the optimization of a SP model. This approach is considered highly relevant for real life scenarios since it produce good results in different combinations of VRP variants such as the Capacitated VRP, Asymmetric VRP, Open VRP and others.

Subramanian et al. [40] propose a hybrid algorithm for a VRP with homogeneous fleet. The developed algorithm makes use of the interaction between a *Mixed Integer Programming* (MIP) solver and an *Iterated Local Search* (ILS) based heuristic to solve an SP formulation of the VRP mainly focusing on flexibility in order to address more than one VRP variant. The ILS heuristic makes it possible to separate the generation of feasible routes from the creation of the final solution. When instances of different VRP variants need to be solved, only the ILS module must be adapted to generate routes that follow the new constraints. The main problem of this kind of strategy resides in solving the SP efficiently. The adopted solution by Subramanian et al. lies in the choice of the dimension of the input set, i.e. the number of routes. The strategy that they choose to make that decision is based over the total number of customers and the average number of customers per route.

The most used strategy in the literature still remains the Column Generation with pricing. A good example is set by Cavaliere et al. [10]. The algorithm developed is able to solve large scale instances of different variants of the VRP. The Local Search heuristic is based over the Lin and Kerninghan [28] heuristic (LK) originally developed for the TSP and extended to tackle many VRP variants. The Column Generation phase uses an LP relaxation of the SP formulation applied to a restricted set of routes filtered from a core set generated beforehand by the local search phase. Computing the reduced costs of the variables of the problem, the best theoretical routes are extracted and included in the Mixed Integer Problem (MIP) formulation of the SP.

2.2 Decomposition methods for large instances

To speed up the execution of VRP algorithms, many use decomposition techniques that help to make the problem easier. Decomposition techniques are used when the instances of the problem are too large and there is the need to split it in smaller sub-problems. This way the initial VRP can be considered like many small TSPs, easier to manage and to solve.

In 1993, Ryan et al. [36] study the Petal Method heuristic that uses a special petal-like structure for route generation, with the depot at the center and the routes that space around it. This decomposition technique has been proved very efficient for large instances with the property of having a central depot surrounded by the customers. This work also proves that a more generalized petal approach for route generation can lead to optimal VRP solutions. The idea of generating more structured and intuitive routes is widely used in practice and a similar approach is also used in the algorithm developed for this thesis real VRP instance.

In the state-of-the-art, the most used decomposition techniques for the VRP are based on customer partitions. Customer partitions allow for different decomposition approaches that consider the geometric position of the requests, e.g. geographically close customers, their temporal aspects such as their Time Windows, or even previous solution attributes and route structure. Early works consider the partitions of vehicles across the possible routes of the solution as a decomposition strategy, but it results less efficient for various VRP variants (such as the VRPTW) compared to customer partitions.

As an example, the work of Taillard et al. [41] is applied to VRP instances with many available vehicles. It introduces a decomposition strategy based on the shortest paths between customers. The main idea is that customers that are near to each other are probably good candidates to be in the same route. This method can be generalized to non-Euclidean problems and works well with different distance based units.

Vidal et. al. [44] in 2013 proposes a simple decomposition framework that partitions the customers in different subproblems to solve separately based over their relative positions. In their work they present a structural and geometrical decomposition technique that addresses large scale instances more efficiently. For smaller instances, the work of Goeke et al. [21] describes a new decomposition method based over already generated routes. Instead of focusing on customers alone, the clustering strategy proposed generates the subproblems considering single routes and their closest routes as candidate sets of customers. Their work demonstrates how the use of some decomposition methods result useful even with medium instances.

Temporal aspects are used by Shaw [38] in 1997 to define the relatedness of two requests based over their Time Windows. Later, the same strategy was used by Ropke and Pisinger [35] to solve a Pickup and Delivery VRP with Time Windows. To choose which request can be effectively visited after another, the possible time of visit is a valuable criterion in such a choice. Close customers with similar TWs are considered good candidates to lie sequentially in the same route.

The importance of decomposition techniques in solving large VRP instances is also addressed by the work of Santini et al. [37] for the INFORMS Journal on Computing. They prove that the use of such strategies generally improve the solution quality regardless of the applied algorithm. Focusing on the most used heuristics, one of the best performing route-based decomposition technique is found to be barycenter clustering, obtaining excellent results on both *Adaptive Large Neighborhood Search* (ALNS) and *Hybrid Genetic Search* (HCG). It's also essential to point out the importance of parameter tuning. In almost every paper about decomposition clustering, it is mentioned that even if a decomposition strategy performs really well on a given instance, without the proper value for each parameter, it can actually result in a worse solution than when no clustering is used.

Chapter 3

Vehicle Routing Problem

3.1 Arc-based VRP formulation

The following model formulation is based on the one proposed by Laporte, Mercure, and Nobert [25] and then used by Toth and Vigo [43].

The VRP can be defined as a complete directed graph $G = (V, A)$. The vertex set V represents the *depot* as node 0, and all the customers, i.e. the Transport Requests. Each arc (i, j) of the set $A = \{(i, j) : i, j \in V, i \neq j\}$ represents a path that a vehicle can take to visit node j departing from node i. It is important to highlight the fact that the graph is directed, meaning that $(i, j) \neq (j, i)$ as shown in Figure 3.1, and that each pair of nodes has two arcs that connects them.

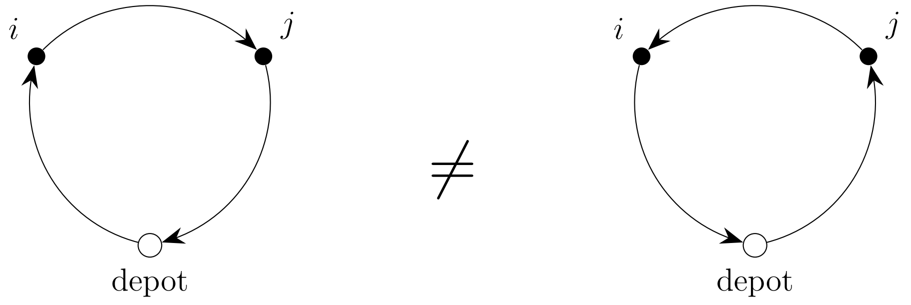


Figure 3.1: Different routes with the same visited customers.

Each arc has an associated cost of crossing c_{ij} that can indicate travel distances, travel time or any other metric cost that must be evaluated. In the original version of the problem, the costs for moving between two points are symmetric, i.e. the graph $G = (V, A)$ is complete and undirected with arcs $(i, j) = (j, i)$ and cost $c_{ij} = c_{ji}$.

The choice of a subset of arcs in sequence from depot to depot models a route, as can be seen in Figure 3.2. We formally define a *route* as a sequence $r =$

$(i_0, i_1, i_2, ..., i_s, i_{s+1})$ with $i_0 = i_{s+1}$ as depot of departure and arrival, and the set $S = \{i_1, i_2, ..., i_s\}$ as the customers served by the route r. The cost of a route is the sum of all the costs associated to the arcs crossed, that is c$(r) = \sum_{p=0}^{s} c_{i_p, i_{p+1}}$. The computation of the route cost can change depending on the VRP variant of choice. There could be additional external costs or costs depending on the properties of the route itself, or even penalties or prizes depending on the customer served. The cost of a route usually reflect the distance that a vehicle needs to travel in order to visit all the customers, or could depend on the time it takes to do so, or the fuel costs plus the driver wage.

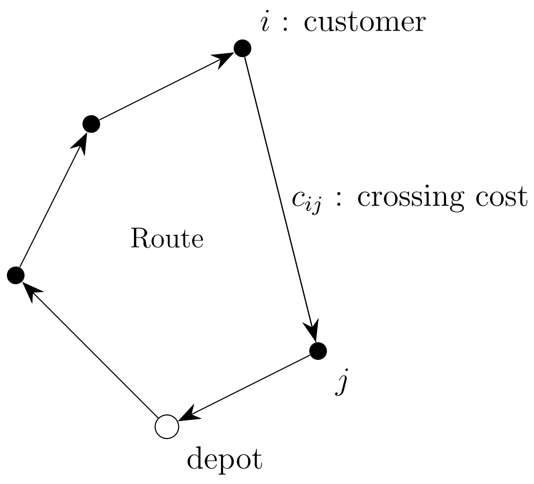


Figure 3.2: Example of route with crossing cost on the arcs.

Assuming we have a fleet K of available vehicles, the objective is to determine a set of minimal cost routes of size $< |K|$ that meet all customer demands. This set of feasible routes generates the *solution* to the VRP.

Let $S \subseteq V/\{0\}$, for directed graphs we define the *in-arcs* and *out-arcs* of S as $\delta^-(S) = \{(i,j) \in A : i \notin S, j \in S\}$ and $\delta^+(S) = \{(i,j) \in A : i \in S, j \notin S\}$. It is standard practice to also define $\delta^+(i) := \delta^+(\{i\})$ for $S = \{i\}$. It follows the mathematical model of the VRP: We define the decision variable x_{ij}:

$$x_{ij} = \begin{cases} 1 & (i,j) \text{ is part of the solution} \\ 0 & \text{otherwise} \end{cases}$$

The two index vehicle flow formulation for the VRP is the following:

$$
\begin{aligned}
\min \quad & \sum_{i\in V}\sum_{j\in V} c_{ij}x_{ij} && && (3.1a)\\
\text{s.t.} \quad & \sum_{j\in\delta^+(i)} x_{ij} = 1 && \forall i \in V/\left\{0\right\} && (3.1b)\\
& \sum_{i\in\delta^-(j)} x_{ij} = 1 && \forall j \in V/\left\{0\right\} && (3.1c)\\
& \sum_{j\in\delta^+(0)} x_{0j} \leq |K| && && (3.1d)\\
& \sum_{(i,j)\in\delta^+(S)} x_{ij} \geq \mathrm{r}(S) && \forall S \subseteq V/\left\{0\right\}, S \neq \emptyset && (3.1e)\\
& x_{ij} \in \{0,1\} && \forall (i,j) \in A && (3.1f)
\end{aligned}
$$

The objective function (3.1a) must minimize the total routing costs. Constraints (3.1b) and (3.1c) represent the *flow conservation constraint* that ensures that every customer is visited exactly once. For each node $i \in C$ the solution must have exactly one arc entering the node and one arc exiting it, meaning also that there is a predecessor and a successor.

Constraint (3.1d) introduces the set K of vehicles and ensures that there are at most $|K|$ arcs exiting the depot, meaning that the solution can be served by the available fleet of vehicles.

Equation (3.1e) represents the *capacity constraint* as well as the *Subtour Elimination Constraint* (SEC). It imposes that the routes must be connected and that the total demand of the customers served per route must not exceed the vehicle capacity. The value $\mathrm{r}(S)$ denotes the minimum number of vehicles required to serve the set $S \subseteq V/\left\{0\right\}$. In the CVRP variant, where each customer demands a certain quantity and each vehicle has a limited capacity, $\mathrm{r}(S)$ can be computed by solving the *bin packing problem* (Martello and Toth [29]) or using the lower bound $\lceil \mathrm{q}(S)/Q \rceil$, with $\mathrm{q}(S)$ the total weight of the customers in S and Q the bins size.

To summarize, a feasible solution for the general VRP formulation has the following properties:

1. each route must visit vertex 0, i.e. the depot, as the first and last vertex of the sequence

2. each vertex $i \in V/\left\{0\right\}$ is visited exactly by one route

3. the final solution must not exceed the fleet size

3.2 VRP Taxonomy

This section aims to cover some of the most important VRP variants in order to introduce the constraints that will be considered in the real life problem addressed in this thesis. It is not in any way an exhaustive VRP taxonomy and does not pretend to be one.

In more than 50 years of history (Laporte [24]), the literature presented many variants of the VRP, each with one or more constraints added to the classical mathematical model. The growth rate of the VRP literature called for the publications of many taxonomies (such as Braekers et al. [5], Toth and Vigo [43] and Eksioglu et al. [17]) that identify and classify any existing VRP variants. The need for a way to consolidate more efficiently all the differences and similarities, the relationships and practical applications of each routing problem, led to the use of a classification based on the single constraint result effect.

The most common variants studied in the literature are described next:

Asymmetric cost matrix VRP **(AVRP)** For all $i, j \in V$, the cost for going from i to j is different from going from j to i. This makes it so that the problem can be modeled using a directed graph. In the literature it's more common to find variants that use symmetric costs, while the AVRP is more suitable for real life scenarios. AVRP can be used to better model cases such as urban areas and services such as mail delivery and garbage collection.

Pickup and Delivery VRP **(PDVRP)** Unlike the classic VRP, the customers can demand both delivery and pickup actions when visited. In more complex variants the capacity of a vehicle must be checked at each vertex since the transported goods must not exceed it at any point in a route. The pickup demand can be returned to the depot [46] or delivered to another customer. A real life scenario could be public transportation.

Split Delivery VRP **(SDVRP)** This variant can be seen as a relaxation of the classic VRP. It states that a single request can be split between different vehicles (Dror and Trudeau [15, 16]). In traditional VRP a customer request must be served by a single vehicle, however in SDVRP it can be served by multiple routes. This variant can be useful when a demand exceeds a vehicle capacity or when splitting it leads to reduced costs.

Multiple Depots VRP **(MDVRP)** When a company has more than one depot to serve its customers, a route can start and end at different depots. There can be different vehicles at each depot due to limited capacity, or each depot can serve different customers (Renaud et al. [33]).

***Open VRP* (OVRP)** This variant can be used when the vehicles are not required to end their route at a depot after covering the last request of the route, or the final trip to the depot is not counted on the costs (Golden et al. [27]).

***Capacitated VRP* (CVRP)** The most common VRP variant, it adds the constraint that each vehicle has a certain capacity, meaning that can transport a certain quantity of goods. It is usually expressed in pallets, kg and m^3.

***Distance-Constrained VRP* (DCVRP)** Route length can often be added as a global constraint as in Laporte et al. [25]. It is usually applied to distance, but it can also be applied to duration, routing costs, or number of stops in a route. This variant introduces an upper bound to the length of a route in the sense that assuming $d_{ij} > 0$ as the distance between vertex i to j, and $L > 0$ as an upper bound on the spatial distance, for each route $r = (i_0, i_1, i_2, ..., i_s, i_{s+1})$ we have the new constraint $\sum_{p=0}^{s} d_{i_p,i_{p+1}} \leq L$.

***Multi-Trip VRP* (MTVRP)** Considering the daily planning of a routing problem, a vehicle could end their planned route way before the working day ends. Allowing a vehicle to perform multiple trips during a given time frame can result in improved efficiency and cheapest solutions. This variant is often applied to real life scenarios and city logistics. Good examples can be found in Battarra et al. [4], Cattaruzza et al. [9] and Mingozzi et al. [30].

***VRP with Time Windows* (VRPTW)** Cordeau et al. [12] introduces the constraints of Time Windows. Each vertex $i \in V/\{0\}$ can have multiple *Time Windows* (TWs) $[a_i, b_i]$, i.e. time frames within which it can be visited. When a route starts, time must be accounted for and properties such as *travel, service, and waiting/idle times* can be introduced in the problem. Travel time is the duration of crossing an arc. Idle times may occur when a vehicle arrives at vertex i before time a_i, hence when its TW has not started yet. Service time can be defined as the time it takes for the delivery service once the vehicle arrives at a customer location, e.g. parking and unloading, and is essential to compute arrival and departure times for each visit. When a vertex i is visited, the variables recording the arrival time t_i^{ARR}, the start of the service t_i^{SER} and the departure time t_i^{DEP} are updated.

Heterogeneous Fleet VRP **(HFVRP)** An heterogeneous fleet (Baldacci et al. [2]) consists of vehicles that differ in capacity, variable and fixed costs, speed and accessible areas, i.e. visitable customers. Fixed costs are those costs that cover a driver salary and vehicle maintenance. A heterogeneous fleet K can be partitioned in $|P|$ subsets of homogeneous vehicles, that is with the same properties, and can be defined as $K = K^1 \cup K^2 \cup ... \cup K^{|P|}$. Fleet size can also be part of the variant, being unlimited or limited, to better suit real case scenarios. Often in real life cases the transport service company can also use multiple fleets, making available its own or relying on third parties.

VRP with Profits **(VRPP)** When a limited fleet is used, it can lead to the impossibility of covering every request of the problem. Only a subset of requests can be covered and a strategy for choosing which ones must be adopted. To control the selection process a prize is added at each customer and it will be removed from the final cost of the route. Note that when negative prizes are used, they become penalities.

3.3 Set Covering Formulation

The Set Covering (SC) formulation of the VRP problem can help solve different VRP variants. The objective of the SC formulation is to determine the best combination of (feasible) routes that partition all customers, minimizing the overall cost.

The integration of the SC formulation in the VRP framework can help address the complexities of the constraints of the VRP variant that needs to be solved. The SC only needs to select a subset of pre generated (feasible) routes, without the need to consider the specific constraints of the VRP variant. This way only the route generation algorithm must handle the specific constraints of the VRP variant.

Let Ω be a set of (feasible) routes of the VRP, and $V/\{0\}$ be the set of Transport Requests to satisfy. We denote as c_r the cost of the route $r \in \Omega$, $|K|$ as the number of available vehicles and a_i^r is a binary variable that asserts if the request $i \in V/\{0\}$ is covered by route $r \in \Omega$.

We define the decision variable θ_r as such:

$$\theta_r = \begin{cases} 1 & \text{route } r \text{ is part of the solution} \\ 0 & \text{otherwise} \end{cases}$$

Then the SC for the general VRP can be formulated as follows:

$$\begin{aligned}
\min \quad & \sum_{r\in\Omega} c_r\theta_r & & \text{(3.2a)}\\
\text{s.t.} \quad & \sum_{r\in\Omega} \theta_r \leq |K| & & \text{(3.2b)}\\
& \sum_{r\in\Omega} a_i^r\theta_r \geq 1 & \forall i \in V/\{0\} \quad & \text{(3.2c)}\\
& \theta_r \in \{0,1\} & \forall r \in \Omega \quad & \text{(3.2d)}
\end{aligned}$$

The objective function (3.2a) aims to minimize the total solution cost required by the selected routes. Constraint (3.2b) ensures that the number of chosen routes doesn't exceed the number of available vehicles, and constraint (3.2c) grants that all customers are served by at least one route. This implies that the solution returned by solving the SC problem is not a feasible solution for the initial VRP and that it still needs to be manipulated in order to make sure that each customer is covered by exactly one of the selected routes.

This formulation can be easily modified to satisfy all VRP variants with different fleet and vehicle constraints. The main advantage of using the SC problem formulation to solve the VRP is the modularity it can give to any algorithm, splitting the route construction and the solution search in two distinct steps that can be customized independently. This lead to the possibility of easier algorithm customization to suite different VRP variants and rich problems. The SC formulation allows for more flexibility and can efficient handle different objective functions without the need of rewriting most of the code from scratch, unlike classic heuristics.

Chapter 4

Current Problem Formulation

The VRP variant used as case study in this paper can be considered an intersection of some of the most popular families of optimization problems studied in the literature.

The considered VRP is characterized by the presence of a single depot from which all vehicles depart and must return to daily. Not all customers must be served even if the main goal remains full coverage. The demand of a request cannot be split between vehicles.

More generally, the considered VRP include the following popular variants:

- Capacitated VRP (CVRP)
- VRP with Time Windows (VRPTW)
- Distance-Constrained VRP (DCVRP)
- Vehicle Routing Problem with Profits (VRPPP)
- Asymmetric cost matrix VRP (AVRP)
- Heterogeneous fleet VRP (HVRP)

These specific features of the problem are directly translated into constraints to be included in the mathematical model. Some other smaller constraints are considered to be able to fully transpose the problem to a real life study case, such as working and driving hours, or incompatibilities between vehicles and customers to be served.

4.1 Constraints

The real life case study of this thesis introduces some highly specific constraints that must be satisfied in order to find feasible routes and solve the problem. As an extra characterization of the problem as a real life VRP, we introduce the objects *Point of Interests* (PoIs). Each customer has a corresponding PoI indicating its geographical location described by its coordinates `(coordX,coordY)`. In the arc-based formulation it can be identified as a node and can be associated to one or more Transport Request.

The problem constraints are described below:

Capacity Similarly to the CVRP variant, each Vehicle has a limited capacity of goods that it can transport. The resource capacity of each Vehicle consists of three different values corresponding to its maximum capacity expressed in pallets, kg and m^3.

Trip Length Following the DCVRP variant, each Vehicle has a maximum distance that it can travel per route, including the final travel to the depot.

Number of Stops per Route Similarly to the Trip Length constraint, each vehicle can have a limited amount of stops per route, meaning that it can visit a limited number of PoIs per route. Consecutive visits to the same PoI counts as one stop only.

Time Windows Each Transport Request can have multiple Time Windows (TWs) that must be respected in order to visit the corresponding PoI and correctly process the operation of delivery. Each Transport Request i has an associated set W_i of TWs. Each TW is defined (like in the VRPTW variant) by the range $[a_t, b_t]^w$, with $0 < w \leq |W_i|$ for indexing. If a vehicle arrives at a PoI to serve a Transport Request i before the start of any TWs in W_i, the rest period is considered idle time for the Working Hours. The service starts when the correct TW opens for the considered Transport Request. For each $i \in V/\{0\}$ we define:

- t_i^{ARR}: arrival time
- t_i^{SER}: service start
- t_i^{DEP}: departure time

Figure 4.1 shows how a simple route can result feasible even if a Time Window is not fully respected. When a vehicle visits a Transport Request before the

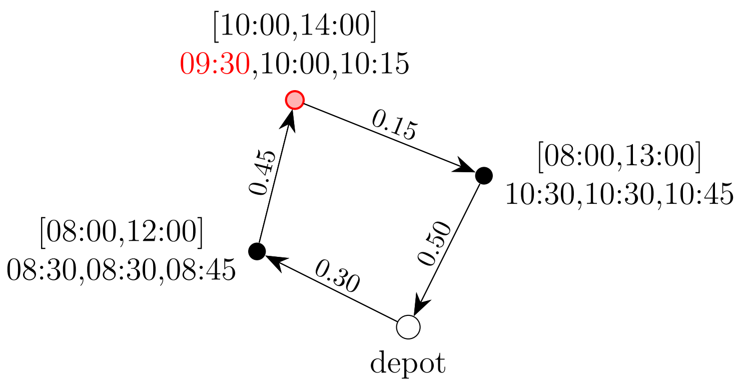


Figure 4.1: Example of route with Time Windows.

start of its Time Window (the node in red), the service start ($t_i^{\text{SER}} = 10{:}00$) will differ from the arrival time ($t_i^{\text{ARR}} = 09{:}30$). This lost time is accounted as idle time and still counts for the final duration of the route. A TW is respected if the service starts within its bounds. No constraint exists for the end time of the service, that can in fact exceed the end of the TW.

Working Hours For this specific study case, the rules applied to regulate the workload of the drivers are bound to the maximum working duration and the maximum driving duration. The working duration is computed as the difference between the arrival time at the depot after the last visit, and the initial departure from it, including wait time and service time. The driving duration is the sum of the driving time between PoIs in the route, excluding idle time before the service start.

Incompatibilities This kind of constraints are included as specific matrices that impose a restriction over the feasibility of a route. The incompatibilities can be found as follows:

- between pairs of PoIs
- between pairs of Transport Requests
- between Transport Requests and Vehicles
- between Vehicles and PoIs

These constraints translate to real life problem such as vehicle incompatibilities with certain roads or locations (e.g. large vehicles can't access small road or the city center) or requests about goods that can be transported only by vehicles with certain properties (e.g. frozen products that need a refrigerated storage).

4.2 Objective Function

The goal of optimization problems is to find the optimal solution, that it is the best solution among all feasible solutions. The optimality of a solution can vary based on the definition of the optimization problem. Given that every solution has a cost, there exists maximization problems that need to find the solution of higher costs, and there exists minimization problems, that need to find the solution of minimum costs.

The VRP is defined as a minimization problem that aims to find the solution of minimum cost.

We consider a feasible solution to be a set of distinct routes that cover up to every Transport Request without using the same vehicle twice. Each route $r = (i_0, i_1, i_2, \dots, i_s, i_{s+1})^k$ is an ordered sequence of Transport Requests with $i_0 = i_{s+1}$ as depot, that satisfy all the constraints described above. Each route r has an associated vehicle $k \in K$. That means that a feasible solution doesn't need to cover all the Transport Requests of the instance and doesn't need to use all available vehicles. In our case the cost of the solutions is not the only value that assumes a major role in the choice of a solution. Given two solutions to compare, the best between them is the one that covers the most Transport Requests. Only in the case where both cover the same number of requests, the solution cost is compared and minimized.

To summary, our algorithm makes use of a hierarchical function where when in need to compare multiple solutions, it always maximizes the number of covered Transport Requests and then minimizes the cost of the solution.

For simplicity, we will now use the term objective function to refer to the total cost of a solution, that is the sum of the cost of the single routes of which it is composed. The cost components of a route in the objective function are the following.

For each vehicle $k \in K$:

- C_k^{VEH}: fixed cost for using vehicle k.
- C_k^{DIST}: cost per unit of traveled distance.
- C_k^{TIME}: cost for route duration including idle time and service time.

For each Transport Request $i \in C$:

- P_i: prize for covering i.

Given a route $r = (i_0, i_1, i_2, ..., i_s, i_{s+1})^k$, we define the *traveled distance* $\mathrm{d}(r)$ as

$$\mathrm{d}(r) = \sum_{p=0}^{s} d_{i_p, i_{p+1}}$$

with d_{ij} the distance between the PoI of request i from the PoI of request j. Note that if the two Transport Requests lay in the same PoI then $d_{ij} = 0$.

The *total duration* $\mathrm{t}(r)$ includes wait time and service time and can be defined as the difference between the arrival time at the depot i_{s+1} and the departure time from depot i_0. Defining the departure time from i_0 as $t_{i_0}^{\mathrm{DEP}}$ and the arrival time at i_{s+1} as $t_{i_{s+1}}^{\mathrm{ARR}}$, the total duration of the route r can be computed as

$$\mathrm{t}(r) = t_{i_{s+1}}^{\mathrm{ARR}} - t_{i_0}^{\mathrm{DEP}}$$

For each Transport Request i served by route r there is an associated prize given for its coverage that can diminish the value of the objective function. This stratagem mimics the worth of the client and helps the algorithm to choose which requests are more valuable and relevant. The total prize of a route is defined as

$$\mathrm{p}(r) = \sum_{p=1}^{s} P_{i_p}$$

In conclusion, the total cost c_r of a route can be computed as follows:

$$c_r = \underbrace{C_k^{\mathrm{VEH}}}_{\text{fixed cost}} + \underbrace{C_k^{\mathrm{DIST}} \sum_{p=0}^{s} d_{i_p, i_{p+1}}}_{\text{variable distance cost}} + \underbrace{C_k^{\mathrm{TIME}} (t_{i_{s+1}}^{\mathrm{ARR}} - t_{i_0}^{\mathrm{DEP}})}_{\text{variable duration cost}} - \underbrace{\sum_{p=1}^{s} P_{i_p}}_{\text{total·prize}}$$

Chapter 5

Algorithm structure

This section provides the algorithm outline for the real life VRP introduced in Chapter 4. We assume that a feasible solution always exists and that it doesn't need to cover all Transport Requests, nor use all the available vehicles.

The algorithm can be divided into five main steps:

1. *Initialization Phase*: parsing the `xml` input instance file and initializing the problem structure. Additional run specific parameters are parsed from the command line.

2. *Route Generation Phase*: a candidate set of feasible routes is populated running a constructive heuristic together with decomposition strategies.

3. *Matheuristic Phase*: this phase is split into three different modules. The first module is called *Filtering Model* and uses a prizing Column Generation strategy to filter the best routes generated, solving an LP relaxation of the initial SC problem. The second module is called *Infeasibility Model* and revolves around a MIP model formulated to find the Transport Requests that cannot be covered by any solution, violating the SC constraint of coverage. Then finally the SC problem can be correctly solved in the *Set Covering Model* with the filtered routes as input and the right set of requests for the coverage constraint.

4. *Patching Heuristic Phase*: the SC optimization returns some routes that must be re-elaborated into a feasible solution. A Patching Heuristic is applied to obtain a feasible solution. Some simple Local Search operators are used to improve it.

5. *Solution update and iterative step*: the Route Generation, the Matheuristic Phase and the Solution Fixage steps are iterated multiple times during a single run of the algorithm. At each iteration, the solution found is compared to the best one found so far and updated accordingly.

A high level representation of the algorithm developed is given in Algorithm 1.

Algorithm 1 High level pseudocode of the whole algorithm.

```
Input: VRP instance file.
Output: The best feasible solution found.
  INIT()
  $CP \leftarrow \emptyset$
  while * not timeOut * do
      $tCP \leftarrow$ HEUR()
      $CP, S \leftarrow$ MATH($tCP$)
      $S' \leftarrow$ PATCH($S$)
      $Solution \leftarrow$UPDATE($S', Solution$)
  end while
  return $Solution$
```

At each iteration the routes generated by the constructive heuristics are saved in the *temporary Candidate Pool* (tCP). The global *Candidate Pool* (CP) is updated and maintained for the next iteration based over the LP Filtering Phase. The algorithm terminates when a certain time limit is reached.

The five steps introuced above are resumed by the call to the methods `INIT`, `HEUR`, `MATH`, `PATCH` and `UPDATE`.

The call to `INIT` initlializes the problem instance and the problem specific parameters. It is directly correlated to the Initialization Phase. The function `HEUR` starts the Route Generating Phase where the candidate set tCP is populated. The call to `MATH` resumes the Matheuristic Phase where three different models are solved using the FICO Xpress Solver. The unfeasible solution S returned is then fixed within the `PATCH` routine and saved in S' as the final feasible solution of the current iteration. S' is then compared to the current global solution with the call to `UPDATE`.

When the time limit is reached, the best solution found is returned. The output of the algorithm includes a detailed description of the solution and of the single routes of which it is composed, its *Key Performance Indicators* (KPI) and an arc-based graphical representation.

To summarize, the algorithm consists in a certaint number of iterations until a time limit is reach. At each iteration the algorithm calls consecutively the main methods `HEUR`, `MATH` and `PATCH`. Each iteration is linked to the previous one by the call to `UPDATE` and by the global Candidate Pool CP.

5.1 Phase 1: Initialization

The input instance is populated from an `xml` file that follows the nomenclature

```
free_xxxx_POIs-xx-TRs-xx_VEHs-xx_InstanceData.xml
```

summarizing the instance ID and the number of PoIs, Transport Requests and vehicles considered.

It is composed by some nodes corresponding to the main objects of the VRP, following a tree-like structure:

- `pois`: the PoIs of the instance, defined by their coordinates (`coordX` and `coordY`) and their Time Windows (`deliveryTimeWindows`).
- `transportRequests`: the Transport Requests that need to be served. Each one of them has a corresponding PoI to be delivered at (`deliveryPoiId`), the delivery demand quantity (`deliveryDemand`) and the delivery service duration (`deliveryUnitServiceDurationVariable` and `deliveryServiceDurationFixed`).
- `vehicleTypes`: the available vehicles with their corresponding costs of usage (`costFixed`, `costPerDistanceUnit` and `costPerDurationUnit`), the max number of stops it can make in a route (`maxNumberOfStopsInTrip`) and its capacity (`resourceCapacity`).
- `fleets`: the vehicles are grouped in different fleets, mimicking the third party companies that can rent their vehicles. We assume the use of non logic fleet, meaning that each vehicle correspond to a single fleet.
- the cost matrices between PoIs: `travelDistMat` for the distance and `travelTimeMat` for the travel duration.
- incompatibility matrices: boolean matrices to map the incompatibilities of some elements. We find the nodes `incMatPoiPoi`, `incMatRequestRequest`, `incMatVehiclePoi` and `incMatVehicleRequest`.

Some other algorithm specific parameters are parsed from the command line. The implemented arguments are the following:

- `-f`: input file name
- `-d`: directory of the input file
- `-t`: time limit
- `-s`: random seed to use
- `-n`: size of the candidate pool
- `-m`: value for the BigM method of the solver
- `-x`: enable the oemLicensing of the XPress Solver
- `-log`: enable the output of the Xpress logs

All the previous parameters are not mandatory and have a default value if not specified.

5.2 Phase 2: Route generation

At each iteration the temporary pool of routes tCP is generated. Two constructive heuristics are proposed in order to generate routes with different characteristics to compare with the global candidate pool CP. The choice of using simple heuristics instead of highly performant Local Search heuristic with many inter-route operators, lies in the objective that the Route Generation Phase must be fast and scalable. The ALSN algorithm used by Optit already implements a very good alternative for most constructive and refinement heuristics and a possible future approach could be the partial merge of the ALSN with the proposed SC algorithm of this thesis.

5.2.1 Cheapest Insertion Heuristic

The first constructive heuristic is based off one of the best-known heuristics called the Savings algorithm by Clarke and Wright [11]. The main difference is that in the VRP considered in the original Savings algorithm all the vehicles were identical. In our rich VRP a heterogeneous fleet is used and the incompatibility constraints led to the choice of generating one route at a time, starting from the choice of the vehicle as the first step of the heuristic.

The pseudocode in Algorithm 2 gives the main steps behind the implemented Savings variant that we will now call Cheapest Insertion heuristic. The main idea

Algorithm 2 Cheapest Insertion constructive heuristic.

```
Input: VRP instance.
Output: Set of routes that generate a solution.
  while Available vehicles $> 0$ AND Available Transport Requests $> 0$ do
      $*$ pick random vehicle $*$
      $route \leftarrow$ new Route(vehicle)
      $*$ pick random $start$ Transport Request $*$
      $route$.add($start$)
      $*$ compute Polar Cluster $*$
      $*$ compute Depot Neighborhood $*$
      while no more move is available do
          $route \leftarrow$ startInsertion()
      end while
      $tCP \leftarrow tCP \cup route$
  end while
  return $tCP$
```

is to generate a possible solution to the VRP instance by choosing sequentially a random vehicle and constructing a route with it. A feasible random Transport Request is chosen as starting point to initialize the route (*depot* $\rightarrow$ *start* $\rightarrow$ *depot*). It's important that the initialization criterion of the route, by the choice of a vehicle and of a starting point, remains random to help make the algorithm non-deterministic.

Two possible decomposition techniques are applied in order to allow a better route generation based over the ideal petal-like structure. Moreover, the use of a decomposition technique helps to minimize the generation time of each route since that way it is possible to avoid the check of the insertion of every possible request of the instance. The decomposition techniques Polar Clustering and Depot Neighborhood are described next in the Subsection 5.2.3, together with other possible clustering methods.

The real work under the Cheapest Insertion algorithm begins after the clustering of the Transport Requests to consider for the current route generation. The call of `startInsertion` allows only feasible insertions in the route. It permits the insertion of a new Transport Request in the current route if no constraint is violated and chooses the insertion of maximum savings. The best insertion to make is the one with the lowest additional cost to the route.

Consider a certain iteration of the algorithm with r as current route. After a new insertion we obtain a new route r'. The savings given by the current insertion is computed as

$$\mathrm{s}(r') = c_r - c_{r'}$$

with c_r cost of route r and $c_{r'}$ cost of route r'. The algorithm chooses the insertion with maximum savings. All Transport Requests in the current cluster are possible candidates for the next insertion and their visit is tested in every possible position of the current route. The while loop continues until no more possible move can be inserted in the current route and the vehicle is considered covered. Then the route is inserted in the temporary Candidate Pool tCP and a new iteration of the Cheapest Insertion algorithm begins with the choice of a different vehicle, until no vehicle and no Transport Request remain available.

The main disadvantage of the Cheapest Insertion heuristic is that it generates one route at a time, without considering the solution as a whole. Focusing on one single route results in a solution generated without any structure guarantee. This feature may be a great disadvantage for a classic VRP heuristic, but the objective of the Route Generation Phase is to populate the set CP with many routes, and not with solutions, needed to initialize the Set Covering Phase that comes next.

5.2.2 Sequential Insertion Heuristic

The second constructive heuristic proposed tries to differ from the logic of the Cheapest Insertion heuristic by generating multiple routes at the same time. Let's call this heuristic Sequential Insertion, partially based over the work of Penna et al. [31]. The pseudocode can be found in Algorithm 3.

Consider a restricted set of vehicles CV. Each vehicle $v \in CV$ have an assigned $route_v$. At each iteration of the algorithm, while there are still uncovered Transport Requests that can be inserted in the solution, all the routes associated to each $v \in CV$ are analyzed in order to insert a new request in each of them. The requests that must be considered during an iteration are saved in the set NN and follow the Depot Neighborhood decomposition strategy. The only difference is that the version used in the Sequential Insertion heuristic can be considered dynamic. Each time that the Depot Neighborhood is called, it amplifies its radius of clustering, until all the requests of the problem instance are to be considered. For each vehicle in CV, a possible new insertion of minimum cost is evaluated at each iteration. Sequentially, all current routes grow one request at a time. When the vehicles are saturated or when the currently considered requests in NN cannot be inserted in any route, the set of vehicles is updated with a new random one.

Algorithm 3 Sequential Insertion constructive heuristic.

```
Input: VRP instance.
Output: Set of routes that generate a solution.
  $NN \leftarrow \emptyset$
  while uncovered Transport Requests > 0  do
      if $NN = \emptyset$ then
          $NN \leftarrow$ Depot Neighborhood++
      end if
      for $v \in CV$ and $NN \neq \emptyset$ do
          $*$ evaluate the costs g(t) of all $route_v.insert(t) \quad \forall t \in NN$ $*$
          $g^{min} \leftarrow \min\{g(t)|t \in NN\}$
          $route_v \leftarrow$ route associated to $g^{min}$
          $*$ update NN $*$
      end for
      if $!\exists t \in NN$ : can be inserted in an $v \in CV$ then
          $*$ update CV $*$
      end if
  end while
  for $v \in CV$ do
      $tCP \leftarrow tCP \cup route_v$
  end for
  return $tCP$
```

5.2.3 Decomposition Techniques

Polar Clustering

Assuming the depot at the origin of a coordinate system, the polar coordinates (r, θ) of all the PoIs that can be visited by the vehicles can be computed starting from their Cartesian coordinates (x, y). The Cartesian coordinates x and y are converted in $r = \sqrt{x^2 + y^2}$ and then the polar angle θ of every PoI is computed as

$$\theta = \begin{cases} \arccos(\frac{x}{r}) & \text{if } y \geq 0 \text{ and } r \neq 0 \\ -\arccos(\frac{x}{r}) & \text{if } y < 0 \text{ and } r \neq 0 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

This coordinate representation allows the use of the decomposition technique called polar clustering. After choosing the *start* Transport Request, a petal-like structure for a route can be obtained by considering only the PoIs within a certain angle from the *start* node. Figure 5.1 shows how from a given *start* request the polar cluster can be determined using its polar angle.

The angle to compute the polar cluster can vary between some values chosen at

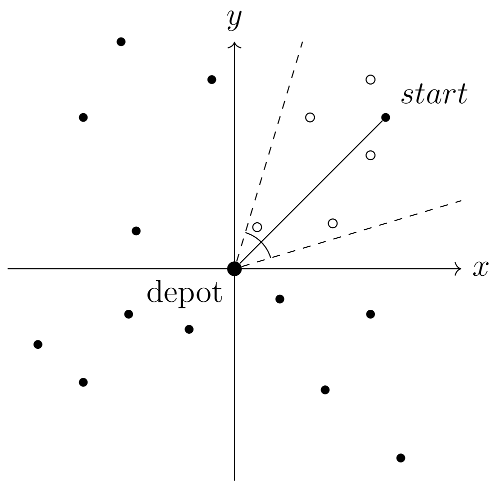


Figure 5.1: Example of Polar Cluster from the *start* node.

random at each iteration to ensure non-determinism. The values considered always remain between $\pm\pi/4$ from the *start* node.

The use of this decomposition technique leads to the creation of routes in the form of a petal-like structure with the depot as the center. In a practical analysis of solutions given as benchmark from the ALNS algorithm of developed by Optit, it has been highlighted how some optimal routes can slightly diverge from this kind of structure, allowing some small requests from around the depot to be inserted in a route, trying to completely saturate a vehicle, i.e. to use all of its capacity. Extending the polar cluster of considered requests at each iteration with a small neighborhood of the depot can help achieve better results.

Depot Neighborhood

A neighborhood of a node is defined as the set of PoIs within a certain distance from it. A simple approach could be to choose a certain radius r and to take all the PoIs within that radius, with the depot as center, as illustrated in Figure 5.2..

To ensure that a minimum number of requests are taken into account, after the radius neighborhood, if the considered Transport Requests are not enough based on a minimum value parameter, the algorithm adds to the cluster the nearest requests until the requested size is met.

The Neighborhood Depot decomposition technique can be used iteratively expanding the radius of the cluster sequentially, exploring a wider area at each call.

Depot Neighborhood and Polar Clustering can be applied together as another decomposition approach (Figure 5.3) based over both distance and polar angle. This different strategy can result in much smaller total travel distance since it groups together closer requests that are probably more related.

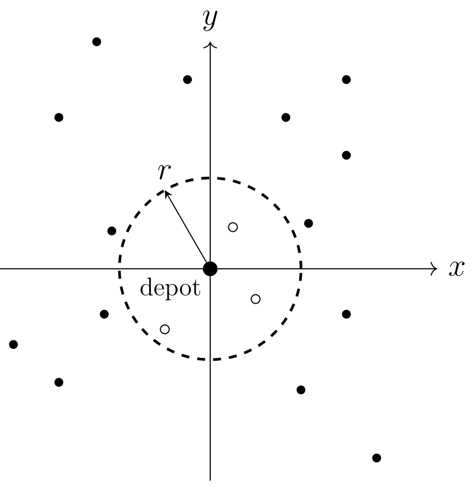


Figure 5.2: Example of Depot Neighborhood.

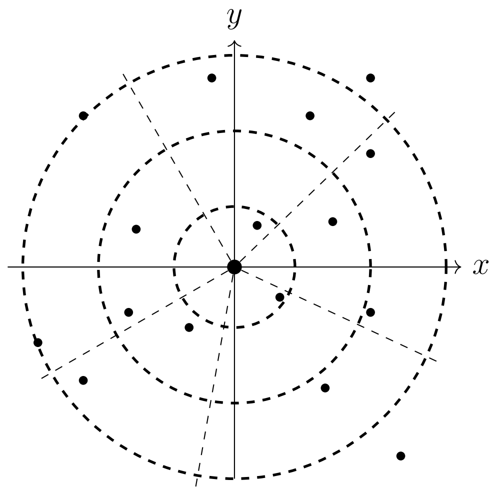


Figure 5.3: Example of Depot Neighborhood and Polar Clustering.

Temporal decomposition

Temporal decomposition, in case of Time Window constrained problems, can result highly efficient to early partition the customers. The main idea behind this approach is the fact that Time Windows can limit the visits to a Transport Requests to a certain period of the day. By logic, requests with similar tight Time Windows won't be visited in the same route unless their corresponding PoI are very close or even the same one.

A vehicle has more probability of visiting early open Transport Requests at the start of a route, rather than at the end of the working shift. The same applies to Transport Requests that have open Time Windows at the end of the day.

A possible use of this time property is the partitioning of Transport Requests based over their Time Windows constraint. In a heuristic algorithm, the choice of a start node can change the effectiveness of the method. Early open Transport Requests can result in a good possible set of nodes to choose from. Requests that have late closing Time Windows can be left as the final visit before the return to the depot.

5.3 Phase 3: Mathematical Approach

5.3.1 Filtering and Column Generation

The filtering Phase follows a similar approach to the one proposed in the paper of Cavaliere et al. [10]. The Set Covering Problem becomes highly impractical to solve when the size of the input set is too large. To exploit this kind of problem a Filtering Phase has been set up. Figure 5.4 shows a simple flow chart describing how the phase works.

The objective of this phase is to find a small and good set of routes from which to start the SC optimization. After the Route Generation phase, the algorithm has two sets of routes from which he must draw the best ones. The temporary Candidate Pool set tCP contains the routes generated during the current iteration, while the global Candidate Pool CP is maintained between iterations and must be updated with the best routes at every call of the Filtering Phase. The CP set has a limited size that can be set during the initialization phase, with a standard value of 15.000 routes.

To select the best routes to maintain in the CP set, the LP relaxation of the SC is solved optimizing both CP and tCP sets. The reduced costs of the routes are computed. The *reduced costs* are defined as the amount by which an objective function coefficient needs to change in order for the corresponding variable to become positive in the optimal solution. It means that when the reduced cost of a route is small, it should be a good candidate for a solution and could possibly be a great variable to consider in the SC optimization.

The new CP set is populated using the routes associated to the variables with the smallest reduced costs. Since the VRP considered uses a heterogeneous fleet, to ensure that all vehicles have an adequate number of possible routes to choose from in the final SC model, the variables and their reduced costs are partitioned in different sets based over the vehicle used. For each set, a subset of routes with the smallest reduced costs is selected. This ensures that every vehicle has a similar number of routes to choose from in the CP for the SC optimization.

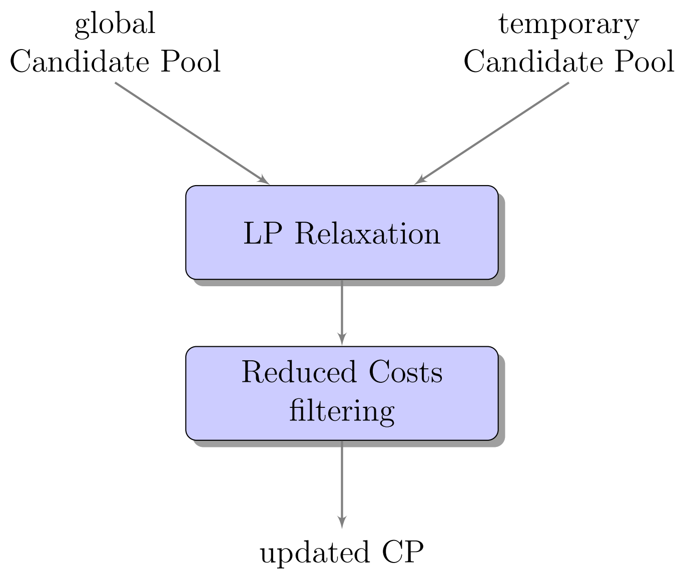


Figure 5.4: Flow Chart of the Filtering Phase.

5.3.2 Set Covering Phase

The main difference between the classic SC formulation for the VRP described in Section 3.3 and the one used for the VRP variant of this thesis, is the use of a heterogeneous fleet where each vehicle $k \in K$ is considered different and unique from one another. In a real life scenario, this property is directly associated to the use of vehicles identified by a license plate.

The new SC formulation doesn't use the general vehicle constraint (3.2b) that ensures that the number of chosen routes doesn't exceed the number of available vehicle. Instead, a new binary variable b^r_k is introduced:

$$b^r_k = \begin{cases} 1 & \text{route } r \text{ uses vehicle } k \\ 0 & \text{otherwise} \end{cases}$$

This variable is used to count how many times a vehicle k is used in the final solution. The new constraint becomes $\sum_{k \in K} b^r_k \theta_r \leq 1$ and replaces the old vehicle constraint in the new SC formulation.

The objective function is also be adapted to account for the prize P_i of each request i served by the solution. We define the variables z_i, $\forall i \in V/\{0\}$:

$$z_i = \begin{cases} 1 & \text{request } i \text{ is served by at least one route} \\ 0 & \text{otherwise} \end{cases}$$

Then the new SC formulation becomes:

$$\min \quad \sum_{r\in\Omega} c_r\theta_r - \underbrace{\sum_{i\in V/\{0\}} P_i z_i}_{\text{prize}} \tag{5.1a}$$

$$\text{s.t.} \quad \underbrace{\sum_{k\in K} b_k^r\theta_r \leq 1}_{\text{new vehicle constr.}} \qquad \forall i \in V/\{0\} \tag{5.1b}$$

$$\sum_{r\in\Omega} a_i^r\theta_r \geq 1 \qquad \forall i \in V/\{0\} \tag{5.1c}$$

$$\theta_r \in \{0,1\} \qquad \forall r \in \Omega \tag{5.1d}$$

Because of the presence of the vehicle constraint (5.1b) of the SC formulation and the incompatibility constraints in the Route Generation, it is not possible to assume that exists a solution that covers all Transport Requests. In addition, the decomposition techniques used in the Route Generating phase don't offer any guarantee over which Transport Requests are served and which not.

The SC constraint (5.1c) imposes that every Transport Request must be covered by at least one route. Since there's the possibility that not all Transport Requests are present in the final solution, this constraint can cause the SC problem to result infeasible. Two different approaches were evaluated to fix this possible situation, each with their positive and negative aspects.

Fake Routes Formulation

To ensure that every Transport Request is covered by the solution, some fake routes are added to the input of the SC problem. In particular, one fake route θ_i is added for every Transport Request $i \in V/\{0\}$, and it uses a fake vehicle k_i that serves only that route and that covers only that request. We define the new vehicle set $K' = K \cup \{k_i : i \in V/\{0\}\}$ to be used in the vehicle constraint (5.1b) instead of K. Each of the new routes is associated to a large cost M to disincentive the solver on choosing that route if not necessary. The SC problem always results feasible with the choice of a fake route that causes a big penalty in the value of the objective function. The Fake Route Formulation also adds the necessity of a subroutine that removes the fake routes from the output solution.

The new formulation makes use of the Big M method to model the new objective function as:

$$\min \sum_{r \in \Omega} c_r \theta_r + \underbrace{\sum_{i \in V/\{0\}} M\theta_i}_{\text{fake routes}} - \sum_{i \in V/\{0\}} P_i z_i$$

The implementation uses the `BIG_M` parameter of the FICO Xpress Solver. By documentation, the "Big M" factor can bring round-off errors in the optimization process and must be tuned according to the other costs in the objective function.

In addition to this, the SC formulation results redundant and the new fake routes make the problem heavier to solve. Different values of the `BIG_M` parameter result in different route choices because of the round-off errors it brings.

Two-Model Formulation

To avoid slow down and the addition of artificial columns in the problem, another possible approach involves the deployment of a second, easier to solve, model identifying the Transport Requests that cause the original SC to be infeasible.

After the Filtering Phase, the new updated CP is given in input to another mathematical model (Figure 5.5), which solution correspond to the Transport Requests that cannot be covered by any possible solution.

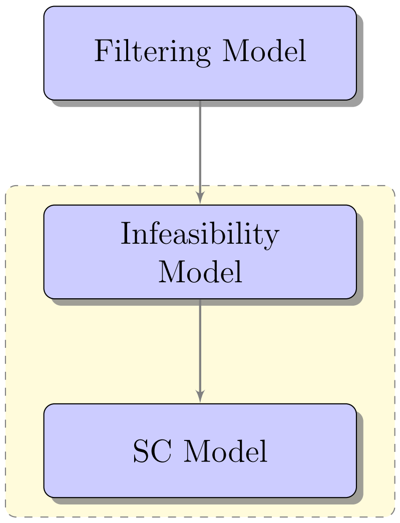


Figure 5.5: Flow Chart of the developed Matheuristic.

We introduce the decision variable y_i, $\forall i \in V/\{0\}$:

$$y_i = \begin{cases} 1 & \text{request } i \text{ cannot be served by any route in the solution} \\ 0 & \text{otherwise} \end{cases}$$

The objective of the new model is to find the Transport Requests that would cause the general SC model to result infeasible. Those requests are then removed from the SC constraint (5.1c).

The new intermediate model called Infeasibility Model is formulated as:

$$\begin{aligned}
\min \quad & \sum_{i \in V/\{0\}} y_i && && \text{(5.2a)}\\
\text{s.t.} \quad & \sum_{k \in K} b_k^r \theta_r \leq 1 && \forall i \in V/\{0\} && \text{(5.2b)}\\
& \sum_{r \in \Omega} a_i^r \theta_r + y_i \geq 1 && \forall i \in V/\{0\} && \text{(5.2c)}\\
& \theta_r \in \{0,1\} && \forall r \in \Omega && \text{(5.2d)}\\
& y_i \geq 0 && \forall i \in V/\{0\} && \text{(5.2e)}
\end{aligned}$$

It is solved by the commercial MIP solver of FICO Xpress and the result is processed to model the SC formulation already introduced without the need of the variable z_i. The objective function (5.2a) aims to minimize the number of requests that cannot be covered. The modified SC constraint (5.2c) allows the variable y_i to take the value of 1 when the request i is not covered by any route with $\theta_r = 1$, avoiding infeasibility reaching the minimum bound (≥ 1).

5.4 Phase 4: Patching Heuristic

After the SC model is solved, an infeasible solution to the VRP is returned. The Set Covering Problem mainly differs from the Set Partitioning Problem by the number of routes in which a certain request can appear. The solution given by the SC optimization can present some Transport Request served multiple times. By removing the redundant request, the final solution cost diminish if done properly.

A Patching Heuristic is developed to remove the redundant requests optimizing the final solution cost. For each route with a multi covered request, a new possible set of routes is generated by removing the excess request. The new possible solutions are analyzed in order to choose from which routes to remove the excess request, and which route to maintain unaltered. The minimum cost solution is chosen.

An example of a possible multi covered request situation (in red) after the SC optimization is shown in figure 5.6. The red crosses mimic the application of the Patching Heuristic to fix the solution, removing the excess request from one route, optimizing the overall costs.

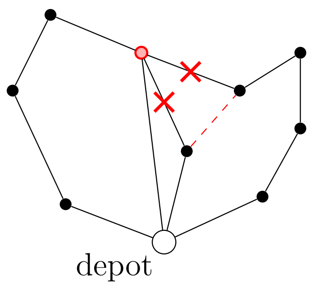


Figure 5.6: Example of a possible patching step for the SC solution.

A simple local search phase is developed in order to reinsert the requests removed from the SC model with the Two-Model Formulation. The operator `REINSERT` works by testing the possible insertion of each uncovered request within each route in the solution until a minimal feasible insertion is found.

After that another basic local search operator is applied. The operator `RELOCATE` is developed to further optimize the routes. The choice of using only simple local search operators is made to show how they can still improve the solution, without the need of an extensive use of more aggressive heuristics.

The `RELOCATE` operator falls in the inter route structure category. It takes a Transport Request i currently belonging to route r and removes it. Then it tries to reinsert it in any other route in the current solution. Let's consider that the current solution is composed by n routes. The `RELOCATE` routine compares n different solutions and maintain the one with minimum cost. The `RELOCATE` operator is called multiple times until no more improving moves are possible. If there are still uncovered requests after the first local search phase, the code iterates again starting from the `REINSERT` operator.

5.5 Phase 5: Solution Update

At each iteration of the program, a new feasible solution S' is generated. If a previous solution S is present, it must be compared to the current one. The algorithm considers different factors to decide which is the best solution between the two.

- The coverage of the solution: the number of Transport Requests served.
- The cost of the objective function, i.e. the total cost of the solution.

Assume we have two solutions S and S'. The first criterion implies that S is better than S' if it serves more Transport Requests. When the total number

```
Number of Routes: 27
Number of Requests: 250
Total Distance: 4451037
Total Cost: 9172.77
Total Prize: 0
Objective Function: 9172.77

Mean Sauration: [ 9.25926%, 67.6333%, 77.9168% ]

Vehicle: 0
Capacity: 100 3500 10000
Coverage: [5%, 82.2571%, 67.18%]

Vehicle: 1
Capacity: 100 3500 10000
Coverage: [5%, 90.4%, 18.11%]

...
...
```

Figure 5.7: Example of KPI console log.

of visited requests is the same, then the cost of the solutions is compared. The minimum cost solution is saved for the next iteration of the program.

The algorithm continues to iterate over the three phases (Route Generation, Filtering and Patching) until a time limit is reached.

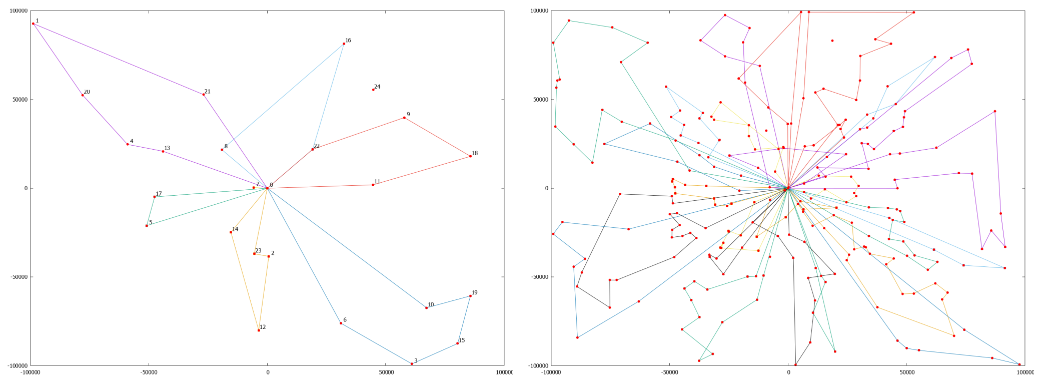
After the run is over, it can be useful to analyze the solution that the algorithm gives in output based over some *Key Performance Indicators* (KPI). The main factors to look for are the coverage of the solution, i.e. the number of covered requests, the number of routes in the solution and the saturation of each vehicle used. The KPI of the solution are shown in the console log (Figure 5.7) and also printed in a text file together with more detailed information about the composition of each route (Figure 5.8).

Because of the inner nature of the measure unit of each coverage value, it is not possible to obtain a high coverage percentage for all three of them. In a good solution of the problem, given a route covered by a certain vehicle, to assert that the vehicle has saturated, only one of the three measures will almost reach the 100% of coverage. Therefore, the mean saturation values computed are never going to reach the full saturation percentage.

```
    ...
    ...


Vehicle: 0
Capacity: 100 3500 10000
Coverage: [5%, 92.8571%, 53.89%]

  Action |  POI | Index     Order | ArrTime   Service   DepTime |       Consumption |     Dist |  Time |
   BREAK |    0 |   250 |         |   21600     21600     21600 |  5   3250   5389 |        0 |     0 |
DELIVERY |   94 |    59 |      59 |   24734     36000     36420 |  4   2333   3516 |    47019 |  3134 |
DELIVERY |   94 |    60 |      60 |   36420     36420     36540 |  3   1795   2485 |    47019 |  3134 |
DELIVERY |   38 |   139 |     139 |   38011     38011     38371 |  2    998   1983 |    69089 |  4605 |
DELIVERY |   93 |    23 |      23 |   39467     39467     39887 |  1    784    714 |    85532 |  5701 |
DELIVERY |   93 |    84 |      84 |   39887     39887     39947 |  0      0      0 |    85532 |  5701 |
   BREAK |    0 |   251 |         |   43453     43453     43453 |  0      0      0 |   138126 |  9207 |

    ...
    ...
```

Figure 5.8: Example of a detailed solution printout.

The graphical plot of the solution is also generated using the software GNUPLOT [45] version 5.4 patch level 6. For the graphical representation of the solution, an arc-based formulation is used similar to the one introduced before. The only difference lies in the definition of the set of vertices V that in the VRP definition represented the Transport Requests. For simplicity, the new set of vertices that are plotted correspond to the geographical position of each request, meaning that each node plotted correspond to a real PoI defined by its coordinates. The depot always lies at coordinate (0,0).



(a) Instance of 50 Transport Requests.

(b) Instance of 250 Transport Requests.

Figure 5.9: Example of solution plot using the GNUPLOT software.

Figure 5.9a is an example of a possible plot of the solution of a VRP instance of 50 Transport Requests, 51 PoIs and 10 available vehicles. In this case, the exceeding PoI refers to the central depot. As can be clearly seen, the solution is composed of 8 routes, meaning that 8 vehicles out of 10 were used. This solution highlight the petal like structure mentioned before. The same structure is also found in the solution of larger instances, as can be seen in Figure 5.9b.

Chapter 6

Results

This chapter presents the results obtained from running the presented algorithm over an artificial set of instances generated by Optit to test its own ALNS algorithm. Some of the tables that are described in this section are too large and were therefore put in the appendix.

The implementation of the algorithm described in this paper was coded in C++17. Tuning and testing were performed on a Computer with an Intel®Core™i7-8550U 1.8GHz CPU and 12 GB DDR4 of RAM. The commercial solver used to model and optimize the Linear Programming problems is FICO Xpress v9.2.2 [18].

6.1 Parameter Tuning

The algorithm uses some parameters that needed to be tuned beforehand in order to perform better for the generation of the final results. The instances used to tune these parameters are a subset of the entire set of instances provided by Optit and do not include the instances used for the final benchmark.

The instances considered can be differentiated by the number of POIs (Point of Interests), Transport Requests and vehicles. Table 6.1 describes how four different categories of instances were considered based on those parameters. Instance IDs refers to the name of the instance file, then NPoI is the number of PoI of the instance, NReq the number of Transport Requests to serve and NVeh the number of available vehicles.

The tables reported next show the results of the tuning of the global parameter Candidate Pool Size and the decomposition parameters Polar Angle and Depot Neighborhood Size.

Instance IDs	NPoI	NReq	NVeh
free_0006-0010	51	50	10
free_0016-0020	100	250	30
free_0026-0030	251	250	50
free_0036-0040	501	500	60

Table 6.1: The four subset of instances for tuning.

6.1.1 Candidate Pool Size

At each iteration of the algorithm, a temporary Candidate Pool tCP of 10000 routes is populated by the Route Generation Phase. The Set Covering problem is solved over the global Candidate Pool CP that is updated at every iteration with new routes. Since the SC optimization performance depends on how many columns the problem has in input, the size of SC must be tuned accordingly. For the set of smaller instances of 50 Transport Requests (from `free_0006` to `free_0010`) the time limit of each run has been set to 1 minutes, with the tunes values of CP size of 2000,3000 and 5000 routes. For the other three sets, because of the higher number of requests to visit, the time limit has been set to 10 minutes each. The instances of 250 Transport Requests (from `free_0016` to `free_0020` and from `free_0026` to `free_0030`) were tested with CP size of 5000, 10000 and 15000. The last set of instances with 500 Transport Requests (from `free_0036` to `free_0040`) has been tested with CP size of 10000, 15000 and 20000.

Table 6.2 shows the results of the tuning of the CP size. The table columns can be read as follows:

CP: the size of the Candidate Pool set.

NR: the number of routes in the final solution

NReq: the number of Transport Requests served by the solution

ObjF: the value of the Objective Function of the solution

	CP	NR	NReq	ObjF	CP	NR	NReq	ObjF	CP	NR	NReq	ObjF
free_0006	2000	7	49	3207.66	3000	7	49	3123.54	5000	7	49	3123.54
free_0007	2000	8	50	3290.74	3000	8	50	3290.74	5000	8	50	3290.74
free_0008	2000	8	50	3386.21	3000	7	50	3146.03	5000	7	50	3146.03
free_0009	2000	8	50	3268.68	3000	8	50	3268.68	5000	8	50	3268.68
free_0010	2000	5	42	2446.15	3000	5	42	2446.15	5000	5	42	2446.15
free_0016	5000	25	250	7966.43	10000	25	250	8061.35	15000	25	250	8023.37
free_0017	5000	25	250	7724.37	10000	26	250	8020.62	15000	27	250	8114.36
free_0018	5000	19	250	6415.9	10000	19	250	6506.78	15000	20	250	6750.35
free_0019	5000	30	240	10650.7	10000	29	240	11022.1	15000	28	239	10917.9
free_0020	5000	18	228	6577.59	10000	18	227	6574.88	15000	18	227	6469.59
free_0026	5000	23	250	9134.23	10000	22	250	9056.57	15000	21	250	8909.44
free_0027	5000	25	250	10096.8	10000	26	250	10307.9	15000	26	250	10273.6
free_0028	5000	25	250	9629.49	10000	23	250	9379.27	15000	24	250	9294.28
free_0029	5000	24	250	9630.07	10000	24	250	9873.9	15000	24	250	9859.42
free_0030	5000	24	250	9612.665	10000	24	250	9826.26	15000	25	250	9737.51
free_0036	10000	60	499	25957.9	15000	60	497	23978.5	20000	60	496	24999.6
free_0037	10000	57	500	22579.1	15000	60	500	23617.6	20000	57	500	21932.1
free_0038	10000	59	500	23327.3	15000	59	500	23460.5	20000	58	500	23495.8
free_0039	10000	54	500	22791.6	15000	57	500	22964	20000	57	500	22964
free_0040	10000	59	500	22454.9	15000	55	499	22919.1	20000	58	500	25082.3

Table 6.2: Candidate Pool size tuning.

Some rows were highlighted to better analyze the results obtained. The cells colored in red show that within a certain run, the number of Transport Requests visited was inferior to the other runs. This behavior must be accounted for the choice of the CP size, since a solution that covers fewer requests is poorer than the others. At the contrary, the cells in green emphasize the good result of the run that covered more requests than the other two. After the tuning, for each set of instances, the CP size value chosen can be read in table 6.3.

Instance IDs	**CP size**
free_0006-0010	3000
free_0016-0020	5000
free_0026-0030	15000
free_0036-0040	10000

Table 6.3: The four subset of instances for tuning.

Unexpectedly, the CP size values don't seem to grow with the number of Transport Requests, nor with the aspect ratio of Requests over available vehicles. This result is probably the direct consequence of using a Set Covering model instead of a more popular Set Partitioning model. The SC problem in fact doesn't give any warranty over the optimization of the feasible route that can be obtained after the Patching Heuristic. After the SC Phase, a more competitive local search phase could be more appropriate and could result more performing.

6.1.2 Decomposition tuning

In the Route Generation Phase two decomposition techniques are called in order to make the algorithm fast and scalable. The Polar Clustering strategy works by taking all Transport Requests within a certain value from the polar angle of a starting requests. Such method is combined with the Depot Neighborhood technique that computes a subset of Transport Requests based on the closeness to the central depot.

The parameters tuned are the value of the angle θ for the Polar Clustering and the size of the Depot Neighborhood to include in the partition. The test were divided by neighborhood size based over the number of Transport Requests of each instance. The values tested for the neighborhood size were 5%, 10% and 20% of the total of the requests, each with the Polar Clustering parameter θ with values 40°, 20° and 30°. The tuning was computed only on the three smaller sets of instances with 50 and 250 Transport Requests.

All the performance profiles were plotted with the use of the software Performance Profile by D. Salvagnin (2016) based over the work of Dolan and Mor [14].

Table 6.4 shows the solution results with Depot Neighborhood size equal to 5% of the number of Transport Requests in each instance. Highlighted in red, the instances `free_0019` and `free_0020` performed slightly worse with $\theta = 30$ and $\theta = 20$ respectively. Looking at the results with the same coverage between runs, it can be seen that the solutions with $\theta = 40$ have the lowest objective function. A similar conclusion can be taken by analyzing the performance profile in Figure 6.1. The tests made with $\theta = 40$ performed clearly better than the other two.

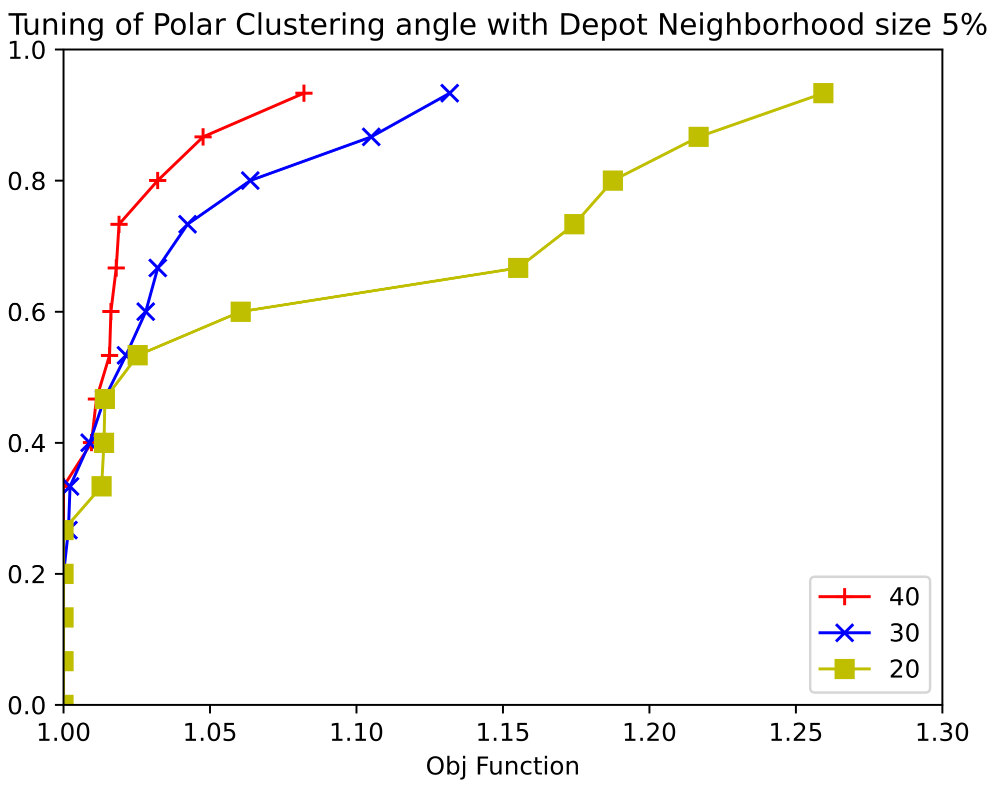


Figure 6.1: Tuning of Polar Clustering angle with Depot Neighborhood size 5%.

Table 6.5 sums up the results of the tuning with a Depot Neighborhood of size equal to 10% of the requests of each instance. The tests with $\theta = 20$ perform way worse than the other two, with less coverage for the instances `free_0006` and `free_0020`. Even though the test with $\theta = 40$ produces a better coverage for instance `free_0019`, by the performance profile in Figure 6.2 can be deducted that the tests with $\theta = 30$ resulted the cheapest overall, followed by $\theta = 40$.

The tuning results with Depot Neighborhood size of 20% of the total requests

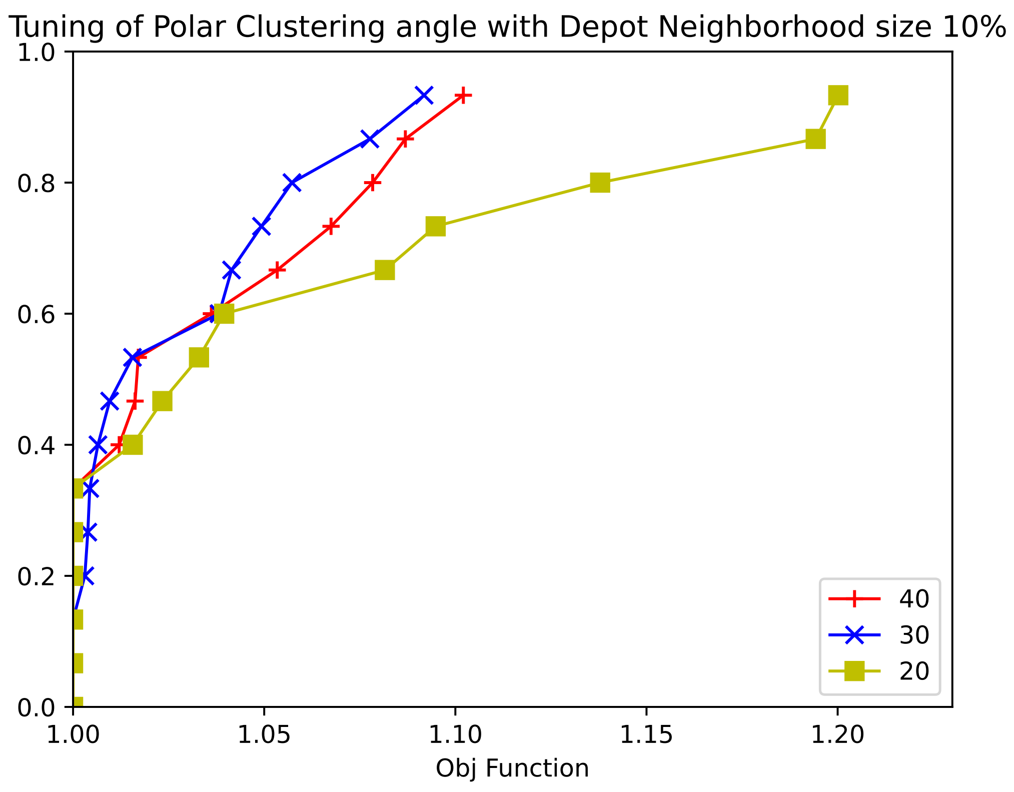


Figure 6.2: Tuning of Polar Clustering angle with Depot Neighborhood size 10%.

are shown in Table 6.6. Both tests with $\theta = 40$ and $\theta = 30$ resulted with better coverage for some instances (highlighted in green). The tests with $\theta = 20$ didn't perform well again. It is now clear how a small angle is not suitable for partitioning the requests to form the popular petal like structure in the solution. Figure 6.3 shows the performance profile of these tests, with the value $\theta = 30$ the best performing.

After the three tuning phases, the solutions that must be compared are the ones produced by using respectively Neighborhood Depot size 5%, 10% and 20% with a Polar Clustering angle of $40°$, $30°$ and $30°$. As can be seen in the performance profile in Figure 6.4, the results with parameters 20% - $30°$ perform the best within the first instances, surpassed only by some solutions obtained with the parameters 5% - $40°$ at last. The final parameter tuning resulted with the choice of the bigger Depot Neighborhood of 20% the size of the Transport Requests set, meaning that in fact the requests around the depot can help generate the cheapest routes, while the Polar Clustering angle parameter should not be too small nor too big, with the choice of $\theta = 30$.

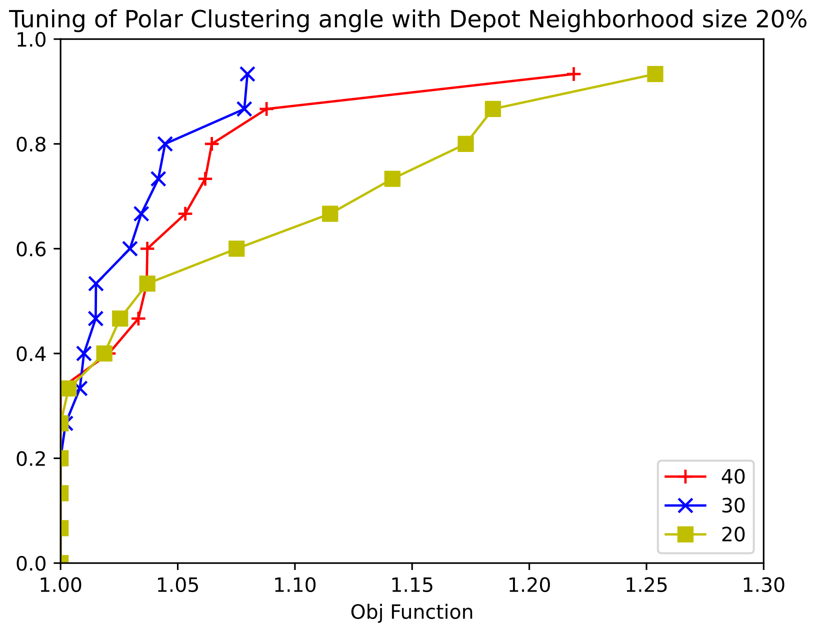


Figure 6.3: Tuning of Polar Clustering angle with Depot Neighborhood size 20%.

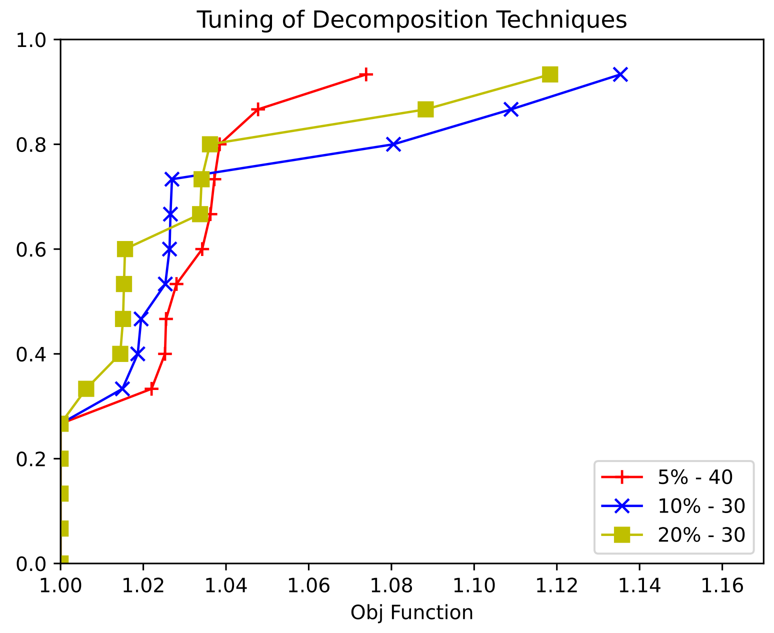


Figure 6.4: Final Tuning of Polar Clustering and Depot Neighborhood.

	θ	NR	NReq	ObjF	θ	NR	NReq	ObjF	θ	NR	NReq	ObjF
free_0006	40	8	49	3272.8	30	7	49	3123.89	20	9	49	3801.56
free_0007	40	7	50	3241.74	30	9	50	3448.51	20	9	50	3807.38
free_0008	40	7	50	3165.34	30	8	50	3582.89	20	9	50	3986.88
free_0009	40	8	50	3219.78	30	8	50	3323.43	20	10	50	3823.94
free_0010	40	6	42	2627.03	30	7	42	2903.17	20	6	42	3035.05
free_0016	40	26	250	8081.14	30	25	250	8023.37	20	25	250	7952.46
free_0017	40	27	250	8003.11	30	27	250	8114.36	20	27	250	8205.73
free_0018	40	20	250	6799.71	30	20	250	6750.35	20	20	250	6735.41
free_0019	40	28	240	10474.3	30	28	239	10917.9	20	29	240	10610.4
free_0020	40	18	227	6571.23	30	18	227	6469.59	20	19	226	6861.05
free_0026	40	22	250	9404.88	30	21	250	9229.98	20	22	250	9357.58
free_0027	40	27	250	10388.9	30	26	250	10273.6	20	26	250	10418.7
free_0028	40	25	250	9639.03	30	26	250	9537.94	20	24	250	9338.84
free_0029	40	26	250	10426.9	30	24	250	9906.42	20	24	250	9635.94
free_0030	40	25	250	10012.1	30	25	250	9851.58	20	26	250	9835.09

Table 6.4: Polar Clustering tuning with Depot Neighborhood at 5%.

	θ	**NR**	**NReq**	**ObjF**	θ	**NR**	**NReq**	**ObjF**	θ	**NR**	**NReq**	**ObjF**
free_0006	40	8	49	3240.72	30	8	49	3375.16	20	9	48	3505.19
free_0007	40	7	50	3170.86	30	8	50	3327.21	20	10	50	3805.95
free_0008	40	7	50	3257.31	30	8	50	3510.36	20	10	50	3890.47
free_0009	40	8	50	3184.79	30	8	50	3306.56	20	8	50	3487.05
free_0010	40	6	42	2627.03	30	7	42	2777.57	20	6	42	2989.5
free_0016	40	25	250	8369.78	30	25	250	7791.33	20	24	250	7761.36
free_0017	40	26	250	8006.83	30	26	250	7911.34	20	27	250	8034.89
free_0018	40	25	250	7167.28	30	19	250	6614.23	20	19	250	6593.94
free_0019	40	30	240	11316.3	30	30	239	10669.8	20	29	239	10600.8
free_0020	40	18	227	6347	30	18	227	6407.75	20	19	226	6598.01
free_0026	40	23	250	9478.98	30	23	250	9327.66	20	21	250	9545.56
free_0027	40	26	250	10461	30	26	250	10330.4	20	25	250	10285.5
free_0028	40	25	250	9783.2	30	24	250	9431.45	20	24	250	9287.08
free_0029	40	26	250	10414.3	30	26	250	10317.2	20	23	250	9449.39
free_0030	40	26	250	10030.8	30	23	250	9680.52	20	25	250	9999.46

Table 6.5: Polar Clustering tuning with Depot Neighborhood at 10%.

	θ	NR	NReq	ObjF	θ	NR	NReq	ObjF	θ	NR	NReq	ObjF
free_0006	40	8	50	3807.94	30	7	49	3123.54	20	8	48	3483.01
free_0007	40	7	50	3150.18	30	8	50	3290.74	20	8	50	3596.49
free_0008	40	7	50	3278.7	30	8	50	3540.27	20	10	50	3845.88
free_0009	40	8	50	3202.48	30	8	50	3336.17	20	9	50	3793.63
free_0010	40	5	42	2426.15	30	5	42	2446.15	20	7	42	3042.25
free_0016	40	26	250	8122.58	30	26	250	7910.82	20	25	250	7832.62
free_0017	40	26	250	7865.07	30	26	250	7706.51	20	26	250	7902.33
free_0018	40	20	250	6822.81	30	20	250	6837.46	20	21	250	6846.38
free_0019	40	30	241	11606	30	30	240	11399.2	20	30	240	11019.5
free_0020	40	18	226	6410.77	30	18	227	6507.65	20	18	226	6648.15
free_0026	40	23	250	9421.8	30	23	250	8873.93	20	20	250	9540.63
free_0027	40	26	250	10505.7	30	25	250	10133	20	25	250	10322.2
free_0028	40	26	250	9811.96	30	24	250	9489.9	20	23	250	9216.98
free_0029	40	25	250	10150.8	30	24	250	10062.3	20	23	250	9330.52
free_0030	40	25	250	10190.6	30	25	250	10010.7	20	25	250	9862.91

Table 6.6: Polar Clustering tuning with Depot Neighborhood at 20%.

6.2 Benchmark comparison

The solved instances by the ALNS algorithm that were made available by Optit are four sets divided by the characteristics of the number of PoIs, number of Transport Requests and number of available vehicles. Table 6.7 describes the instances that we are going to compare next and with which tuned parameters the algorithm developed has solved them. The column CPsize refers to the size of the Candidate Pool, θ the Polar Clustering angle parameter and DNsize the size of the Depot Neighborhood (20% of the Transport Requests set size).

Instance IDs	**NPoI**	**NReq**	**NVeh**	**CPsize**	θ	**DNsize**
free_0001-0005	25	50	10	3000	30	10
free_0021-0025	251	250	30	15000	30	50
free_0031-0035	100	500	60	10000	30	100
free_0031-0035	200	1000	100	15000	30	200

Table 6.7: The four subset of instances for tuning.

Table 6.8 compares the benchmark solution values of Optit, generated by their ALSN algorithm with time limit of 300 seconds and a maximum of 1000000, with the results of the SC algorithm proposed in this thesis. The SC algorithm is run with a time limit of 15 minutes per instance. The choice of having a wider time limit is to bring closer the gap of performance between the two algorithms. It is essential to remember that the SC algorithm doesn't have the goal to surpass the ALSN algorithm, instead it should be ideal to merge them trying to improve the weaknesses of both.

The table is divided in two main columns representing the ALSN solutions (left) and our algorithm solutions (right).

Looking at the last two columns of the table, with the values of the number of Transport Requests covered and the cost of the objective function, some cells are highlighted in red, yellow and green.

Firstly, more evidence is made for the NReq column, since a solution with more coverage results always better without the consideration of the solution cost.

The green instances show more coverage, meaning that the decomposition techniques used were highly performing, while other instances like `free_0035` or `free_0048` ,these same techniques led to covering much less requests, resulting into too strict partitions. Less coverage is mainly found within the bigger instances of 500 and 1000 Transport Requests. Those instances were not particularly tested during the tuning parameter phase and the decomposition techniques resulted less effective.

	ALSN algorithm			SC algorithm		
	NR	NReq	ObjF	NR	NReq	ObjF
free_0001	5	50	2199.147	7	50	2522.91
free_0002	4	50	2134.198	6	50	2410.45
free_0003	7	50	2197.347	9	50	2338.26
free_0004	5	50	2356.434	7	50	2621.28
free_0005	4	50	1993.333	7	50	2549.51
free_0021	22	250	10046.749	26	250	10402.7
free_0022	22	250	10424.536	30	250	11521.1
free_0023	26	250	10887.098	29	250	10634.2
free_0024	23	249	9968.519	25	250	10069
free_0025	25	249	11211.501	29	250	10710.9
free_0031	60	500	19472.044	54	500	16603.9
free_0032	60	499	18981.504	57	498	18825.4
free_0033	57	500	17126.099	55	500	17274.6
free_0034	52	500	17014.202	50	499	16217.7
free_0035	60	500	20391.985	57	495	20139.4
free_0046	100	983	34589.704	100	988	38620.567
free_0047	100	965	39888.517	100	842	52540.2
free_0048	100	989	36276.13	100	915	54577
free_0049	100	994	35631.464	100	942	56679.6
free_0050	100	997	34878.231	98	996	35633.4

Table 6.8: Comparison of the ALSN algorithm with the SC algorithm.

Meanwhile, when the same coverage is met, the smaller instances (from `free_0001` to `free_0005`) resulted in worse costs. The yellow highlights mean that the solution cost found is within the 20% of the cost of the ALSN solution. We say that those results are good for a first approach to an SC algorithm based over the cheapest paths heuristic, but it can be easily improved by adding some more competitive local search.

Overall, the SC algorithm developed resulted in more expensive solutions than the ALSN solutions. The results can be partitioned in three groups: small instances of 50 Transport Requests, medium instances of 250 and 500 Transport Requests and big instances of 1000 Transport Requests.

To sum up, the SC algorithm worked as follows:

Small Instances: although the SC algorithm solutions fully cover the Transport Requests set, the results show a great difference of approximately 300 unit of costs from the ALSN objective function values. One possible cause is due to use of simple constructive heuristic without an extensive use of any local search, narrowing down the number possible routes to generate.

Medium Instances: the SC algorithm exploits the incompatibility constraints by combining different routes with the objective of finding the solution with more coverage. Without the solution feasibility constraint, more routes are merged together where the ALSN algorithm must maintain solution feasibility. The SC algorithm shows more difficulties following the performance of the ALSN algorithm with the growth of the Transport Requests set size.

Big Instances: with the growing number of Transport Requests to combine to generate different routes, the lack of a competitive local search heuristic and the use of too strict decomposition techniques that are not tuned to such instance sizes, lead the SC algorithm to generate too expensive routes.

Chapter 7

Conclusions

In this thesis we have presented a possible matheuristic framework for the solution of a rich VRP, with focus over the use of a Set Covering formulation. The SC Formulation takes in input a large set of routes that are optimized with the objective of finding a minimum cost solution to the VRP.

The algorithm can be described as modular, consisting of a main module that hosts the matheuristic itself, solving three different mathematical models. Two more modules are needed for the algorithm to work as intended: a constructive heuristic needed to generate the input set of routes, and a Patching Heuristic that fixes the SC solution into a feasible VRP solution. Those two modules can be re implemented with any heuristic of choice, making the algorithm fully customizable for many VRP variants that needs to be solved.

The constructive heuristic implemented uses a simple cost based criterion to choose the next best Transport Request insertion in a route, merging the shortest path method with time and distance unit costs. The use of simple heuristics maintains the algorithm fast and scalable, giving more focus on the matheuristic module in the middle.

Some light decomposition techniques are used based on customer partitioning. In particular, Polar Clustering that takes advantage of the natural petal like structure of the routes, and Depot Neighborhood that exploits the closeness to the depot. The choice of using both helps the algorithm to be faster and better performing. Some light Local Search is also added to the Patching Heuristic applied at the end of the SC optimization in order to demonstrate how the solution can be easily further improved.

The results obtained from the SC algorithm show how it can be used to exploit the common weaknesses of other Local Search heuristics, like the ALSN heuristic used by Optit. The SC algorithm works well with multiple inter-route constraints since the use of the SC formulation results more relaxed than the sequential generation of feasible solution of the common Local Search heuristics.

With large instances, the SC algorithm clearly performs worse because of the need of a more focused constructive heuristic and more competitive decomposition techniques. The lack of a real Local Search phase after the Matheuristic Phase is also a major weakness.

Overall, the SC algorithm proposed has definitely potential for improvements. As a possible future work, the first hypothesis to test is to change the constructive heuristic with a more popular one from the literature, fast and scalable, to generate good routes for the SC model input set. Different criteria can be used other than the euclidean distance and time duration to generate routes with different underlying principles. Many different decomposition techniques can be tested, starting from time based partitioning or even vehicle partitioning.

The Patching Heuristic can also be improved by adding more Local Search operators that works both intra-route and inter-route, exploiting both Route Generation Phase and Mathematical Phase as a single constructive heuristic to work upon. In fact, the second main idea is to merge the SC algorithm with the ALSN algorithm by Optit, trying to cover the weakness of both algorithms.

All-in-all, the SC algorithm demonstrated great promise for complex medium instances, being easily adaptable to many real life VRPs. The focus on route combination rather than solution feasibility helps the algorithm to find solution with full coverage, other than the minimization of costs. Being fully customizable, the SC algorithm result highly adaptable to real life logistic problems, making the study of the SC problem attractive for future development of the field.

References

[1] AGCOM. Osservatorio sulle comunicazioni, 2/2023.

[2] R. Baldacci, M. Battarra, and D. Vigo. Routing a heterogeneous fleet of vehicles. *The vehicle routing problem: latest advances and new challenges*, pages 3–27, 2008.

[3] M. L. Balinski and R. E. Quandt. On an integer program for a delivery problem. *Operations research*, 12(2):300–304, 1964.

[4] M. Battarra, M. Monaci, and D. Vigo. An adaptive guidance approach for the heuristic solution of a minimum multiple trip vehicle routing problem. *Computers & Operations Research*, 36(11):3041–3050, 2009.

[5] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300–313, 2016.

[6] J. Caceres-Cruz, P. Arias, D. Guimarans, D. Riera, and A. A. Juan. Rich vehicle routing problem: Survey. *ACM Computing Surveys (CSUR)*, 47(2):1–28, 2014.

[7] A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Operations research*, 47(5):730–743, 1999.

[8] D. Cattaruzza, N. Absi, and D. Feillet. Vehicle routing problems with multiple trips. *4or*, 14:223–259, 2016.

[9] D. Cattaruzza, N. Absi, D. Feillet, and T. Vidal. A memetic algorithm for the multi trip vehicle routing problem. *European Journal of Operational Research*, 236(3):833–848, 2014.

[10] F. Cavaliere, E. Bendotti, and M. Fischetti. An integrated local-search/set-partitioning refinement heuristic for the capacitated vehicle routing problem. *Mathematical Programming Computation*, 14(4):749–779, 2022.

[11] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

[12] J. F. Cordeau, G. Desaulniers, J. Desrosiers, M. M. Solomon, and F. Soumis. VRP with time windows. *in The Vehicle Routing Problem, P. Toth and D. Vigo*, ch.7:155–194, 2002.

[13] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[14] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[15] M. Dror and P. Trudeau. Savings by split delivery routing. *Transportation Science*, 23(2):141–145, 1989.

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

[17] B. Eksioglu, A. V. Vural, and A. Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472–1483, 2009.

[18] FICO. Xpress v9.2.2 optimization. `https://www.fico.com/en/products/fico-xpress-optimization`, 2023.

[19] B. A. Foster and D. M. Ryan. An integer programming approach to the vehicle scheduling problem. *Journal of the Operational Research Society*, 27:367–384, 1976.

[20] R. De Franceschi, M. Fischetti, and P. Toth. A new ilp-based refinement heuristic for vehicle routing problems. *Mathematical Programming*, 105:471–499, 2006.

[21] D. Goeke, T. Gschwind, and M. Schneider. Upper and lower bounds for the vehicle-routing problem with private fleet and common carrier. *Discrete Applied Mathematics*, 264:43–61, 2019.

[22] B. L. Golden, S. Raghavan, E. A. Wasil, et al. *The vehicle routing problem: latest advances and new challenges*, volume 43. Springer, 2008.

[23] J. P. Kelly and J. Xu. A set-partitioning-based heuristic for the vehicle routing problem. *INFORMS Journal on Computing*, 11(2):161–172, 1999.

[24] G. Laporte. Fifty years of vehicle routing. *Transportation science*, 43(4):408–416, 2009.

[25] G. Laporte, M. Desrochers, and Y. Nobert. Two exact algorithms for the distance-constrained vehicle routing problem. *Networks*, 14(1):161–172, 1984.

[26] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[27] F. Li, B. Golden, and E. Wasil. The open vehicle routing problem: Algorithms, large-scale test problems, and computational results. *Computers & operations research*, 34(10):2918–2930, 2007.

[28] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[29] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[30] A. Mingozzi, R. Roberti, and P. Toth. An exact algorithm for the multitrip vehicle routing problem. *INFORMS Journal on Computing*, 25(2):193–207, 2013.

[31] P. H. V. Penna, A. Subramanian, and L. S. Ochi. An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, 19(2):201–232, 2013.

[32] D. Pisinger and S. Ropke. Large neighborhood search. *Handbook of metaheuristics*, pages 99–127, 2019.

[33] J. Renaud, G. Laporte, and F. F. Boctor. A tabu search heuristic for the multi-depot vehicle routing problem. *Computers & Operations Research*, 23(3):229–235, 1996.

[34] Y. Rochat and É. D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of heuristics*, 1:147–167, 1995.

[35] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.

[36] D. M. Ryan, C. Hjorring, and F. Glover. Extensions of the petal method for vehicle routeing. *Journal of the Operational Research Society*, 44(3):289–296, 1993.

[37] A. Santini, M. Schneider, T. Vidal, and D. Vigo. Decomposition strategies for vehicle routing heuristics. *INFORMS Journal on Computing*, 2023.

[38] P. Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, 46, 1997.

[39] Optit S.r.l. `https://www.optit.net/`. Accessed: 2024-01-18.

[40] A. Subramanian, E. Uchoa, and Luiz S. Ochi. A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40(10):2519–2531, 2013.

[41] E. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673, 1993.

[42] P. Toth and D. Vigo. An overview of vehicle routing problems. *The vehicle routing problem*, pages 1–26, 2002.

[43] P. Toth and D. Vigo. *Vehicle routing: problems, methods, and applications.* SIAM, 2014.

[44] T. Vidal, Teodor G. Crainic, M. Gendreau, and C. Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & operations research*, 40(1):475–489, 2013.

[45] Thomas W., Colin K., and many others. Gnuplot 5.4 patchlevel 6: an interactive plotting program. `http://gnuplot.info`, 2023.

[46] A. C. Wade and S. Salhi. An investigation into a new class of vehicle routing problem with backhauls. *Omega*, 30(6):479–487, 2002.