CZECH TECHNICAL UNIVERSITY

MASTER THESIS

Multi-constraint Vehicle Routing Problem Solver with GRASP Metaheuristic

Author:
Bc. Jakub LEČBYCH

Supervisor:
Ing. Petr POŠÍK, Ph.D.

A thesis submitted in fulfillment of the requirements
for the degree of Master of Software engineering and technology

in the

Department of Cybernetics

January 2, 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Lečbych** Jméno: **Jakub** Osobní číslo: **457806**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Algoritmus využívající GRASP meta-heuristiku pro řešení problému routování vozidel s omezeními

Název diplomové práce anglicky:

Multi-constraint Vehicle Routing Problem Solver with GRASP Metaheuristic

Pokyny pro vypracování:

* Analyze the types of constraints for Vehicle Routing Problem (VRP) and their effect on the problem solution.
* Survey existing relevant approaches for the solution of the chosen variant of constrained VRP.
* Design a VRP solver based on GRASP meta-heuristic.
* Compare the results of your solver(s) with other available tools (OR-tools, OptaPlanner, jsprit)

Seznam doporučené literatury:

• Greedy Randomized Adaptive Search Procedures: Advances and Extensions (Mauricio G. C. Resende, Celso C. Ribeiro), 2018, DOI: 10.1007/978-3-319-91086-4_6
• Bio-inspired Algorithms for the Vehicle Routing Problem (Francisco Baptista Pereira, Jorge Tavares), 2009, ISBN 978-3-540-85151-6
• Vehicle Routing: Problems, Methods, and Applications (Daniele Vigo, Paolo Toth), 2014, ISBN 1611973589
• Vehicle Routing with Pickup and Delivery: Heuristic and Meta-heuristic Solution Algorithms (Hosny Manar), 2021, ISBN 978-3-659-20258-2
• Constructing initial solutions for the multiple vehicle pickup and delivery problem with time windows (Hosny M., Mumford Ch.), 2021, DOI 10.1016/j.jksuci.2011.10.006

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Petr Pošík, Ph.D., katedra kybernetiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **25.05.2021** Termín odevzdání diplomové práce: **04.01.2022**

Platnost zadání diplomové práce: **19.02.2023**

Ing. Petr Pošík, Ph.D.		prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.

Datum převzetí zadání	Podpis studenta

Declaration of Authorship

I, Bc. Jakub LEČBYCH, declare that this thesis titled, "Multi-constraint Vehicle Routing Problem Solver with GRASP Metaheuristic" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Abstract

Multi-constraint Vehicle Routing Problem Solver with GRASP Metaheuristic

The Vehicle Routing Problem (VRP) is a problem that has been studied in the literature for several decades and involves routing a fleet of cars to service a group of consumers. The Pickup and Delivery Problem (VRPPD) is a well-known version of the VRP. In the VRPPD, it is generally required to find one or more low-cost routes to deliver/pickup goods to/from customers. Another noteworthy version is VRP with Time Windows (VRPTW), where each site is assigned a time window and the aim is to service all customers within that time window. The entire transportation cost for both tasks should be minimized while adhering to a set of pre-specified problem restrictions. Applications of VRP are common in everyday transportation and logistics services, and the issue is anticipated to become much more prominent in the future as e-commerce and Internet shopping become more popular. The real-world cases of VRP cannot be optimally solved in a reasonable time, necessitating the use of heuristic techniques.

The purpose of this study is to examine the GRASP metaheuristic with different construction heuristics that pertain to solving VRP variants. Contrary to previous research in this area, I devote my attention to tackling tough issues and constraints in a straightforward and practical manner. I do so without complicating the process of finding a solution. Experiments have shown that the implemented algorithm is better than other baseline algorithms. Further, my trials have shown that my newly added components can intensify and diversify more effectively than previous components.

In summary, the study findings indicate that the algorithm is successful in dealing with tough issue constraints and developing simple and resilient solution that can be incorporated with vehicle routing management tools and employed in a range of real-world applications.

Key words

Vehicle routing problem, VRP, pickup and delivery, time windows, routing, planning, heuristics, construction heuristics, greedy randomized adaptive search procedure, GRASP

Abstrakt

Algoritmus využívající GRASP meta-heuristiku pro řešení problému routování vozidel s omezeními

Problém plánování tras (VRP) je problém, který je v literatuře studován již několik desetiletí a zahrnuje plánování tras vozidel za účelem obsluhy jednotlivých zákazníků. Varianta problému zahrnující vyzvednutí a doručení (VRPPD) je dobře známá verze VRP. Ve VRPPD je obecně vyžadováno najít jednu nebo více tras, ve kterých se doručí/vyzvedne zboží zákazníkům/od zákazníků. Další verzí je VRP s časovými okny (VRPTW), kde je každé lokalitě přiřazeno časové okno a cílem je obsloužit všechny zákazníky v daném časovém okně. Celkové přepravní náklady u obou variant by měly být minimalizovány při dodržení všech předem specifikovaných omezení. Aplikace VRP jsou běžné v každodenní přepravě a logistických službách a je očekáváno, že tato problematika bude v budoucnu mnohem významnější, díky větší oblibě elektronického obchodování a nakupování přes internet. Protože případy VRP v reálném světě nelze optimálně vyřešit v rozumném čase, je nutné pro řešení použít heuristické metody.

Účelem této studie je prozkoumat GRASP metaheuristiku, s různými variantami konstrukčních heuristik,pro řešení VRP. Na rozdíl od předchozích výzkumů v této oblasti věnuji svou pozornost řešení těžkých problémů a omezení přímým a praktickým způsobem, aniž by proces hledání řešení byl výrazně komplikovaný. Experimenty ukázaly, že implementovaný algoritmus je lepší než jiné základní algoritmy. Dále mé pokusy ukázaly, že mé nově přidané složky mohou zesílit a diverzifikovat poskytnutá řešení účinněji než předchozí složky.

Závěry studie naznačují, že algoritmus je úspěšný při řešení náročných omezení a vývoji jednoduchých a spolehlivých řešení, která lze začlenit do nástrojů pro správu plánování vozidel a použít v řadě aplikací v reálném světě.

Klíčová slova

Problém plánování tras, VRP, vyzvednutí a doručení, časová okna, routování, plánování, omezení, heuristiky, konstrukční heuristiky, GRASP

Acknowledgements

First and foremost, I'd like to thank my thesis supervisor, Ing. Petr POŠÍK, Ph.D. for his patient guidance, counsel, material resources, and criticism during my work on this thesis. His skills and experience were extremely beneficial to both the direction and outcome of our project.

Second, I'd like to take this opportunity to thank all the department faculty members for their help and encouragement throughout my studies. I'd also like to thank David Mokoš, my coworker, for sharing his knowledge and expertise with me.

Contents

Declaration of Authorship **v**

Abstract **vii**

Acknowledgements **xi**

1 Introduction **1**

1.1 Research problem 2
1.2 Scope and purpose 3
1.3 Thesis overview 3

2 Problem definition and variants **5**

2.1 Mathematical formulation 5
2.1.1 Capacitated VRP 6
2.1.2 VRP with time windows 6
2.1.3 VRP with pickup and deliveries 6
2.1.4 VRPPDTW formulation 7
2.2 Other variants of vehicle routing problem 9

3 VRP Heuristics **11**

3.1 Construction heuristics 11
3.2 Related work 12
3.3 Cost function 12
3.3.1 Complexity analysis and implementation issues 13
3.3.2 (Sub)route construction and selection 14
3.4 Routing algorithm 15
3.5 The sequential construction algorithm 17
3.6 The parallel construction algorithm 17
3.6.1 Best route parallel construction 18
3.6.2 Best request parallel construction 18
3.6.3 Parallel insertion k-Regret construction heuristic 19

4 GRASP implementation **21**

4.1 Metaheuristic 21
4.2 Random multi-start procedure 21
4.3 Semi-greedy multi-start procedure 22
4.3.1 Effect of alpha value on semi-greedy construction 22
4.3.2 Parallel vs sequential multi-start procedure 25
4.4 GRASP metaheuristic 25
4.5 GRASP algorithm for VRPPDTW 26
4.5.1 Recap on problem formulation 26
4.5.2 GRASP construction 26
4.5.3 Local search 27

4.5.4 Effect of LS algorithm on solution's quality 29

5 Computational Experimentation 33
5.1 Data-set characteristics 33
5.2 Existing solvers and baseline algorithms 33
5.3 Comparison criteria 33
5.4 Comparing the construction heuristics 34
5.4.1 Implementation issues and complexity analysis 34
5.4.2 Comparing with an exiting insertion heuristic 36
5.4.3 Heuristics as an initial solution for OR-Tools 36
5.5 Comparing the results of GRASP with different construction heuristics 39
5.5.1 Comparing with exiting solvers 42

6 Conclusion 45
6.1 Summary 45
6.2 Future work 45

Bibliography 47

List of Figures

2.1 A example of PDMVRPTW instance. Before solution (left) after solution (right) 5

3.1 Solution construction 12
3.2 Time window selection 14

4.1 Distribution of the solution cost obtained by the semi-greedy construction procedure as a function of α value. (1000 repetitions of a minimization problem of VRP on instances with 100 customers and 10 vehicles) 23
4.2 Average cost of best solution obtained by construction heuristic with different alpha values 24
4.3 Effect of parallelization of the GRASP algorithm with PBR 7 semi-greedy construction procedure 25
4.4 Distribution of solutions for PBR construction heuristic 30
4.5 Distribution of solutions for PBQ construction heuristic 30
4.6 Distribution of solution's cost difference before and after application of LS procedure 31

5.1 Cost evolution of the GRASP algorithm with different construction heuristic 39
5.2 Cost of best solution found by GRASP with PBR and PBQ construction functions with less greedy HC algorithm 4 40
5.3 Cost of best solution found by of GRASP with PBR and PBQ construction functions with more greedy HC algorithm 5 40
5.4 Cost of best solution found by of GRASP with PBR and PBQ construction functions and both versions of HC algorithms 41
5.5 Cost evolution of PBR and PBQ constuction functions with 3 minute time limit on an instances with 100 customers and 10 vehicles 42

List of Tables

2.1 Notation used in the VRP, VRPTW and VRPPD formulations 7

5.1 Result of construction heuristics 35
5.2 Result of existing insertion heuristic with PBR and PBQ 37
5.3 Result of PBR, PBQ a PIR and insertion heuristics as initial solution to OR-tools solver on instances with 300 customers and 50 vehicles 38
5.4 Result of different solvers on instances with 500 customers and 70 vehicles 43

List of Abbreviations

TSP	**T**raveling **S**alesman **P**problem
VRP	**V**ehicle **R**outing **P**problem
CVRP	**C**apacitated **VRP**
VRPPD	**VRP** with **P**ickup and **D**elivery
VRPTW	**VRP** with **T**ime **W**indows
VRPPDTW	**VRP** with **P**ickup and **D**elivery under **T**ime **W**indows
SEQ	**Seq**uential constuction heuristic
PBR	**P**arallel **B**best **R**route constuction heuristic
PBQ	**P**arallel **B**best Re**Q**uest constuction heuristic
PIR	**P**arallel constuction heuristic with **I**nsetion **R**egret
IRV	**I**nsertion **R**egret **V**alue
HC	**H**ill **C**limbing algorithm
GRASP	**G**reedy **R**ansomized **A**daptive **S**earch **P**rocedure
HALNS	**H**ybrid **A**adaptive **L**arge **N**eighbor **S**earch algorithm

Chapter 1

Introduction

In recent years, logistics have been increasing tremendously and play a huge role in our everyday lives. From ordering food to manufacturing or distributing shared bikes and scooters in the cities. All these areas start to rely on efficient planning to save money spent on transportation. In fact, it is not unusual for some companies to spend more than 20 % of the product's value for logistics and product transportation [22]. In addition, the transportation sector itself is a significant industry, and its volume and impact on society as a whole continue to increase every day. Logistics investments in Europe went up to 38.64 billion euros in 2020. This is driven by the massive growth of e-commerce in Europe [35]. The use of automated route planning and scheduling can lead to huge savings in transportation costs, typically ranging from 5% to 20% [16], which should contribute to boosting the overall economic system.

Research in efficient vehicle planning, routing, scheduling, and optimization has increased significantly, thanks to better technology, computational power, and more data. Experimental work with autonomous vehicles and overall progress in the field of machine learning led to new and better algorithms. Numerous techniques have already been successfully implemented in commercial logistics software and applications. However, due to the increasing demands, dynamic settings, and growing complexity of this sector, new innovations and techniques are still needed to optimize vehicle routing, scheduling, and planning [20].

As a result, restaurants often utilize platforms such as Bolt, Wolt, Grap, or Uber to manage their online sales and deliveries. These platforms frequently have hefty fees, which significantly reduce the earnings of the business. Additionally, restaurants with many chains require more sophisticated planning as they have dozens of drivers delivering food simultaneously. GoDeliver[1] is one of the software solutions that aims to solve this challenge. It offers tools for order administration, automatic order dispatching, automatic route planning, real-time courier tracking, and more to businesses. As of now, GoDeliver's planning algorithm is still not suitable for large instances in complicated and dynamic settings with hundreds of customers.

The goal of this thesis is to develop an efficient planning GRASP algorithm to solve complex pickup and delivery vehicle routing instances with time windows. The results will be utilized to improve GoDeliver's existing planning algorithm.

Experimental results of the implemented algorithms show overall better solutions in several metrics than the baseline algorithms. Results indicate that the newly implemented solver is an adequate candidate as a new solver used in the GoDeliver's planning algorithm.

Section 1.1 reviews the literature and other research papers related to this problem and briefly researches three main variants of VRP that the solver can solve: capacitated, time window, and pickup and delivery. Section 1.2 covers the scope and

[1]https://www.godeliver.co

main purpose of this research. The section 1.3 gives an overview of the content of this thesis.

1.1 Research problem

The vehicle routing problem (VRP) aims to find optimal sets of routes in the transportation network for a fleet of vehicles [12]. This NP-hard problem generalizes the classical Traveling Salesman Problem (TSP) a canonical combinatorial optimization problem that has been widely studied in the literature [4] - or more specifically to multiple TSP - by requiring the assignment of a subset of vertices to a vehicle and the sequencing of these vertices to create a feasible solution. Since the introduction of this problem by [1], extensions and other variants of this problem have been created to meet realistic application in the complex and dynamic world.

Many variants of VRP have emerged since the first publication, and the literature and research show a continuous trend towards the study of more complex VRP variants [37]. The following classes of VRPs are often called "rich VRPs.

- VRP with capacity constraints (CVRP), where a vehicle has limited cargo space.
- VRP with pickup and delivery constrain (VRPPD), where a vehicle must visit a depot before it can serve a customer.
- VRP with time windows constrain (VRPTW), where a vehicle can visit a customer in a specified time window. This time window can either be hard (cannot be violated) or soft (the vehicle is penalized if arrives outside of the time window).

More about these VRP classes in chapter 2.

Research [2] have analyzed the complexity of the vehicle routing problem and have concluded that practically all the vehicle routing problems are NP-hard (among them the classical vehicle routing problem) since they are not solved in polynomial time.

According to these surveys and technical reports [10], [19], [18], [16], all extensions of the vehicle routing problem discussed here are NP-hard, thus it makes a strong point for applying heuristics and metaheuristic to solve the problem.

Other approaches have also been explored by [6]. Their dynamic programming algorithm has been used with great success to obtain a solution to the shortest path problem with time window constrain. Enhancement of their algorithm by [8] proved that even huge instances with up to 2,500 nodes and 250,000 arcs are solvable in less than one minute on CYBER 173[2] [11]

Applications of capacitated VRP with pickup and delivery, and time windows (CVRPPDTW) are becoming even more important due to increasing growth of transportation in food delivery and e-commerce sector [34]. Besides this sector, the application of CVRPPD can be used in other sectors such as the air or ship cargo industry [27].

[2]A Cyber 170-series system consists of one or two CPUs that run at either 25 or 40 MHz, and is equipped with 10, 14, 17, or 20 peripheral processors (PP), and up to 24 high-performance channels for high-speed I/O

1.2 Scope and purpose

This thesis addresses the problem of finding a (near) optimal solution for vehicle routing problem (VRP), its corresponding variants using a GRASP (greedy randomized adaptive search procedure) algorithm, and concern to develop an efficient planning algorithm based on the recent research in VRP area to solve a complex pickup and delivery vehicle routing instances with time windows. The main difference to other papers is that the algorithms used in this thesis are constructed in such a way, that they are able to solve most of these variants combined (after applying the corresponding mapping). Particularly in scenarios involving time limits, pickup and delivery, and skill and capacity limitations.

In our situation, we are attempting to solve a logistical challenge in the food sector in which the meal needs to be picked up first but not earlier than the specified pickup time and delivered to the customer within a particular time window. Moreover, the routes that have already been created are not fixed and might alter throughout the day as new orders are received. As a result, the solver has a limited amount of time to create or find a near-optimal solution. In other words, the solver must find a solution (if one exists) within a specified time limit (usually 3-5 minutes). For this use case, we have decided in the GoDeliver's team to create multiple dedicated in-house solvers for our specific problem. The second reason for this decision to create an in-house solver was a growing instance size where the solvers we were using could not find a good enough solution in time. Most of the solutions for instances with more than 100 points (customers) were far from optimal, thus making the plan almost unusable.

The result will be used to improve the current GoDeliver's planning algorithm. Thanks to the in-house solution, future research, and development of last-mile logistics will be faster and easier. New GoDeliver's planning algorithm will be able to solve complex and bigger instances giving us an advantage over other solvers such as OR-tools.

1.3 Thesis overview

Chapter 1 defines the problem of routing and planning in logistics, researches the literature and the evolution of VRP, describes the motivation and purpose behind this thesis, and finally gives an overview of the structure of this thesis.

Chapter 2 provides a mathematical definition of the problem and describes researched variants.

Chapter 3 presents the methodology of construction heuristics, fitness function, and complexity analysis of the construction algorithm.

Chapter 4 presents the methodology of the GRASP algorithm and its corresponding hybridization variants.

Chapter 5 shows the computational experiments of the proposed algorithm. Compare the results between different construction heuristics and between implemented GRASP solver and other solvers.

Chapter 6 summarizes the implemented algorithm and construction heuristics and research undertaken in this thesis. It also provides future work within GoDeliver team.

Chapter 2

Problem definition and variants

As previously mentioned in section 1.1, routing and scheduling represent an important part of many transportation/distribution systems. The vehicle routing problem (VRP) aims to find optimal sets of routes in the transportation network for a fleet of vehicles [12, 21]. This NP-hard problem generalizes the classical Traveling Salesman Problem (TSP) - or more specifically the Multiple TSP - by requiring an assignment of a subset of vertices to each vehicle and sequencing of these vertices to create a feasible solution. Many variants of the generic VRP have been intensively studied in the literature [13, 3, 9]. These variants that mainly differ in the objective function and constraints are reviewed in the following subsections.

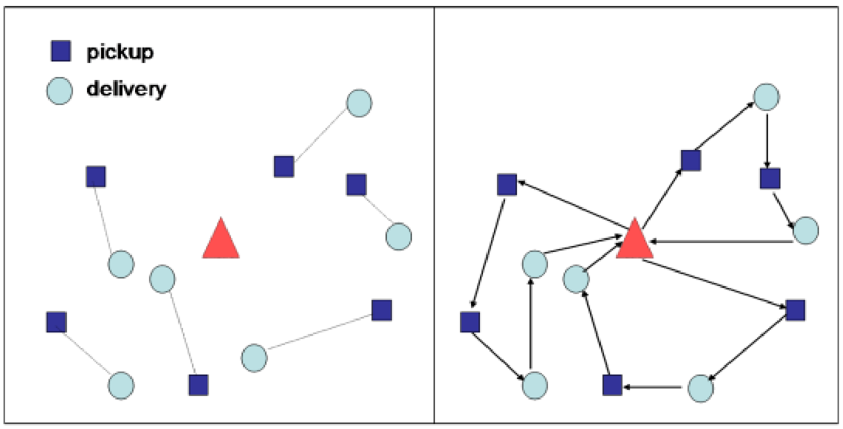


FIGURE 2.1: A example of PDMVRPTW instance. Before solution (left) after solution (right)

2.1 Mathematical formulation

The standard objective function for all VRP problems (VRPPD, VRPTW, CVRP,...) is to minimize the fleet size and/or the sum of travel times and/or the sum of distances traveled with a constraint, that the vehicle must have enough capacity for transporting the goods between nodes.

I assume a complete digraph[1] $G = (N, A)$ with set of nodes N and set of arcs $A = \{(i, j) : N \times N, i \neq j\}$. Note that $|A| = n(n-1)$ where $n = |N|$. Notations are summarized in **Table 2.1**. The problem formulation and constraints are described hereafter (Section 2.1.4). [32]

[1] At least one of the pair of vertices $i, j \in N$ has asymmetric cost $c_{ij} \neq c_{ji}$.

2.1.1 Capacitated VRP

The Capacitated Vehicle routing problem (CVRP) is the most studied version of VRP, although it has primarily an academic relevance [27]. In CVRP, the problem introduces an additional variable denoted as scalar CAP. The goal is to deliver goods by the homogeneous[2] fleet of vehicles $V = \{1, 2, 3, ..., |V|\}$ with capacity $CAP > 0$, from one depot - at node 0 - to a set of customers $N = \{1, 2, 3, ..., n\}$. Each customer $i \in N$ has a demand given by a scalar $q_i \geq 0$, e.g., number of items or weight of the goods. A vehicle serving a subset of customers starts and ends its route in the depot having a route cost equal to the sum of c_{ij}.

2.1.2 VRP with time windows

The VRP with Time Windows (VRPTW) is a generalization of the CVRP involving the added complexity of allowable service time within customer's defined earliest and latest service times [14]. Note that the service times can be either hard time windows (bank deliveries, school bus routing), or soft time windows (food deliveries, e-shop deliveries). In case of hard time windows, if a vehicle arrives at a customer location too early, it must wait till it can begin the service task, due dates cannot be violated. Soft time windows allow a vehicle to arrive outside of specified time windows but the vehicle is penalized by doing so by adding a fixed/dynamic penalty to the vehicle route cost [23]. Soft time window can be encoded as list of hard time windows with different penalties (additional cost added to the final objective function). The problem is defined the same way as in the CVRP with the additional variable t_i representing the beginning of the service time within the time window defined by time interval $\langle e_i, l_i \rangle$. Each node in VRPTW can have multiple time windows in which a vehicle can arrive (e.g., the first interval between 10am-12am and the second interval between 13pm-15pm).

The route (path) in the graph G must satisfy all the constrains defined by CVRP. Each node $i \in N$ has a time window $\langle e_i, l_i \rangle$. A duration c_{ij}^m is associated with each arc $(i, j) \in A$ and time interval $m \in M$. Recall t_i is defined as a start of the service time at node i.

2.1.3 VRP with pickup and deliveries

Similarly as in VRPTW, in the VRP with Pickup and Delivery (VRPPD), a vehicle fleet must satisfy a set of transportation requests. Each request is defined by a pickup point, a corresponding delivery point, and a demand q_i to be transported between these locations [15]. VRPPD involves additional sets of constraints coupling the pickup and corresponding delivery points on the same vehicle routes and visit precedence among all pickup points and their associated delivery points. A simple VRPPD is shown in figure 2.1.

The route (path) in the graph G consist of pickup nodes $i \in P$ and corresponding delivery nodes $j \in D$. It is possible that different pickup/drop nodes may represent same geographical location. Set of pickup points is denoted as $P = \{1, 2, \ldots, n\}$ and set of delivery points is denoted as $D = \{n + 1, \ldots, 2n\}$ and $N = P \cup D$. [25]

[2]All vehicles are identical, i.e., they have the same operating costs and cargo capacity.

2.1.4 VRPPDTW formulation

A combination of all the previously mentioned variants is called VRP with Pickup and Delivery under Time Windows and is formalized by the following equation.

Notation table	
Symbol	**Definition**
N	Set of nodes; In VRPPD $N = P \cup D$
P	Pickup nodes $P = \{1 \ldots n\}$
D	Delivery nodes $D = \{n+1, \ldots, 2n\}$
V	Vehicle set
m	number of time intervals
B	A large constant
c_{ij}^m	Cost (e.g. travel time) from node i to node j at the time interval m
S_i	Service time at node i
CAP	Capacity of the vehicle
q_i	The load at node i
T_{ij}^m	Upper bound for time interval m for link *(i,j)*
e_i	Earliest time that the vehicle can arrive at node i
l_i	Latest time that the vehicle can arrive at node i
t_i^v	The time vehicle v starts service at node i
w_i^v	The load of the vehicle v upon leaving node i
DPT_i	Desired pickup time at node i; $(i \in P)$
DDT_i	Desired delivery time at node i; $(i \in D)$
DRT_i	Desired ride time; $(DRT_i = t_{i+n} - t_i - S_i)$
MRT_i	Maximum ride time
$x_{i,j,v}^m$	Binary variable. If any vehicle v travels from node i to node j during the time interval m, the variable is equal to 1. Otherwise is equal to 0
R	The set of feasible routes satisfying all constrains (2.3) - (2.20)
c_r	Cost of route r
a_{ir}	The number of times a node is visited by route r; $(i \in P)$
y_r	Binary variable. If route r is used in the solution, the variable is equal to 1. Otherwise is equal to 0

TABLE 2.1: Notation used in the VRP, VRPTW and VRPPD formulations

$$\text{minimize} \sum_{i \in N} \sum_{j \in N} \sum_{m \in M} \sum_{v \in V} c_{i,j}^m x_{i,j,v}^m \tag{2.1}$$

$$\text{s.t.} \tag{2.2}$$

$$\sum_{j \in N} \sum_{m \in M} \sum_{v \in V} x_{i,j,v}^m = 1 \qquad \forall i \in P \tag{2.3}$$

$$\sum_{i \in P} \sum_{m \in M} x_{0,i,v}^m = 1 \qquad \forall v \in V \tag{2.4}$$

$$\sum_{j \in D} \sum_{m \in M} x_{j,2n+1,v}^m = 1 \qquad \forall v \in V \tag{2.5}$$

$$\sum_{j \in N} \sum_{m \in M} x_{i,j,v}^m - \sum_{j \in N} \sum_{m \in M} x_{n+i,j,v}^m = 0 \qquad \forall i \in P, \forall v \in V \tag{2.6}$$

$$\sum_{j \in N} \sum_{m \in M} x^m_{j,i,v} - \sum_{j \in N} \sum_{m \in M} x^m_{i,j,v} = 0 \qquad \forall i \in P \cup D, \forall v \in V \tag{2.7}$$

$$t_j \geq t_i + S_i + c^m_{i,j} - B(1 - x^m_{i,j,v}) \qquad \forall i,j \in N, \forall m \in M, \forall v \in V \tag{2.8}$$

$$t_i - T^{m-1}_{i,j} x^m_{i,j,v} \geq 0 \qquad \forall i,j \in N, \forall m \in M, \forall v \in V \tag{2.9}$$

$$t_i + B x^m_{i,j,v} \leq T^m_{i,j} + B \qquad \forall i,j \in N, \forall m \in M, \forall v \in V \tag{2.10}$$

$$e_i \leq t^v_i \leq l_i \qquad \forall i \in N, \forall v \in V \tag{2.11}$$

$$w^v_j \geq w^v_i + q_j - B(1 - \sum_{m \in M} x^m_{i,j,v}) \qquad \forall i,j \in N, \forall v \in V \tag{2.12}$$

$$w^v_i \leq CAP \qquad \forall i \in N, \forall v \in V \tag{2.13}$$

$$x_{i,j,v} \in \{0,1\} \tag{2.14}$$

$$m \in M \tag{2.15}$$

$$v \in V \tag{2.16}$$

$$t^v_i \geq 0 \tag{2.17}$$

$$w^v_i \geq 0 \tag{2.18}$$

$$N \in \{\{0\} \cup \{2n+1\} \cup P \cup D\} \tag{2.19}$$

The formulation consists of the objective function (2.1) that minimizes overall travel costs. And the following constraints:

- Constraint (2.3) guarantees that each demand has to be served once, and each demand is only allowed to be visited by one vehicle.
- Constraints (2.4) and (2.5) ensure that all vehicles must start from the depot and return to the depot.
- Constraint (2.6) ensures that each request (customer) must be picked up first and then delivered with the same vehicle.
- Constraint (2.7) represents the flow conservation equations.
- Constraint (2.8) calculates the departure time to node j.
- Constraints (2.9) and (2.10) are the temporal constraints. If the vehicle travels from node i to node j during time interval m, the departure time of the vehicle from node i is between the upper bound for time interval $m-1$ and upper bound for time interval m.
- Constraint (2.11) imposes the time windows restrictions.
- Constraints (2.12) and (2.13) impose the capacity constraints.
  - Constraint (2.12) is the sub-tour elimination constraints.
  - Constraint (2.13) ensures that the vehicles do not exceed the vehicle capacity limitation.

The time-dependent formulation (VRPTW) is decomposed into the main problem and a set of sub-problems. The main problem becomes the set partitioning problem and the sub-problem becomes the constrained shortest path problem. The main problem determines the optimal feasible vehicle routes based on the meaningful subset of the feasible vehicle routes. The time-dependent VRPPDTW formulation can be reformulated by using path flows instead of link flows. Each route means one vehicle-route (**v**) in the time-dependent VRPPDTW formulation.

$$a_{ir} = \sum_{j \in N} \sum_{m \in M} x^{m}_{i,j,v} \forall i \in P, \forall r \in R, \forall v \in V \tag{2.20}$$

The route cost for each vehicle can be expressed as follows:

$$c_r = \sum_{i \in N} \sum_{j \in N} \sum_{m \in M} c^{m}_{i,j} x^{m}_{i,j,v} \forall r \in R, \forall v \in V \tag{2.21}$$

The mathematical formulation for the main problem is constructed as follows:

$$\text{minimize} \sum_{r \in R} c_r y_r \tag{2.22}$$

$$\text{s.t.} \tag{2.23}$$

$$\sum_{r \in R} a_{ir} y_r = 1 \qquad \forall i \in P \tag{2.24}$$

$$y_r \in \{0,1\} \qquad \forall r \in R \tag{2.25}$$

Where the total cost of the selected route is minimized by the objective function (2.22). Constraint (2.24) ensures that each request (customer) is visited by one vehicle. The objective of the sub-problem is to minimize the total reduced cost for the constrained shortest path problem. An elaborate discussion on this formulation can be found in [28].

2.2 Other variants of vehicle routing problem

It is good to mention other types of VRP. These problems will not be covered by this thesis despite being relevant in real-world applications.

- *Skill based* problem where every vehicle has a set of skills that determines the type of nodes a vehicle can visit.
- *Backhauls* problem where the set of vertices consists of two subsets: linehaul vertices for which the demand is delivered from the depot and backhaul vertices for which the demand is picked up and brought back to the depot.
- *Periodic vehicle routing* problems where routes are determined over a horizon that spans several periods and where each vertex must be visited at a given frequency within this horizon.
- *Inventory routing* where each vertex has an inventory and delivery routes are determined to replenish the inventory to avoid any inventory shortage.
- *Mixed fleet and size* problems where the fleet size must be determined based on different types of vehicles with different characteristics (e.g., capacity).

Chapter 3

VRP Heuristics

There are two techniques to solving VRP problems: exact methods and approximate (heuristic) methods. The exact method will be found to be optimal if enough processing resources are available. Due to the high complexity and computational needs of this technique, only small VRP instances with up to tens of nodes (customers) are optimally solved [2]. Exact methods are frequently unsuitable in real-world scenarios because the complexity and instance size are too huge to deal with in an acceptable period. Heuristics methods, as opposed to exact methods, give a solution with a trade-off between quality and computation time. These heuristic algorithms are chosen over their counterparts even though they have no assurance of producing a high-quality answer. The quality of the solutions provided by these heuristic algorithms can only be determined experimentally through tests and observations. [30, 2, 29].

3.1 Construction heuristics

VRP construction heuristics are algorithms that gradually create a viable solution while reducing the total cost of the solution. There are two major approaches to this problem. The first tries to progressively create the routes by adding a new (unassigned) point to the best route. The alternative strategy is based on the "saving" approach, in which the algorithm gradually combines separate paths, lowering the total solution cost. The Clarke and Wright savings algorithm [33] is a well-known heuristic for VRP that represents the second approach.

Vehicle routing problem with pickup and delivery under time windows (VRPPDTW) is a mix of VRPTW and VRPPD, as discussed in the previous chapter. According to [5], VRPPDTW is known to be NP-hard due to the existence of several restrictions. All varieties of VRP are both grouping problems (assigning the request to the vehicle) and routing problems (finding the best route for each vehicle). A suitable algorithm should be able to handle both sides of the VRP issue effectively.

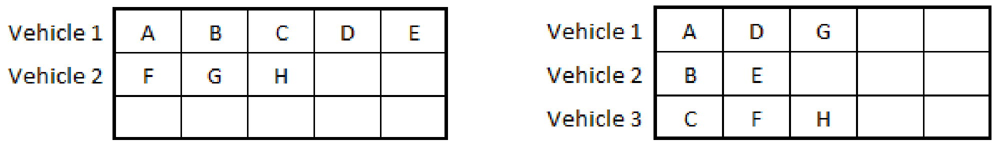
When constructing a solution for VRPPDTW, the algorithm normally selects an unassigned customer request and inserts it into the route that results in the lowest cost overall. The request is inserted into the route at the best (with the lowest cost) feasible position on the particular route. For this type of insertion, an additional calculation is required to determine the impact of insertion on all customers already assigned to that route in terms of delay, feasibility, travel duration, etc. Additional decisions on how to construct the routes include the selection of the customers and the route building process - sequential or parallel. Order of selection of the customers may affect the quality of the final solution. Generally, customers are sorted by proximity to the depot or by time window.

3.2 Related work

The previous section mentions sequential and parallel build approaches for the VRP. The sequential construction process builds the routes one by one, while the parallel construction process builds all routes simultaneously.

Sequential construction (see Fig.3.1a) approach usually adopts the Solomon's sequential insertion heuristics for the VRPTW [7]. This kind of insertion heuristic primarily minimizes the number of vehicles used since the sequential algorithm assigns the request to the first vehicle until the vehicle's route is feasible. Then to the second vehicle, and so on. This may result in assigning no requests to some vehicles.

Parallel construction (see Fig.3.1b) approach inserts requests into any of the available routes. Accordingly, the algorithm requires an initial estimate of the number of routes. If the request cannot be inserted (insertion produces an infeasible solution) to any of the routes, a new route is added and the request is inserted into that route.



Vehicle 1	A	B	C	D	E
Vehicle 2	F	G	H		

(A) sequential construction

Vehicle 1	A	D	G		
Vehicle 2	B	E			
Vehicle 3	C	F	H		

(B) parallel construction

FIGURE 3.1: Solution construction

3.3 Cost function

The most important part of the construction phase of the algorithms is the cost function. A cost function can be any function that accepts the solution and produces a single number or vector of numbers that indicate how good or bad the resulting solution is. As the most often invoked function in the construction algorithm. This function should be fast.

The cost function that measures the quality of the partial solution (route) in the algorithm above is described by the equation 3.1.

$$\begin{aligned} cost(r) = {} & w_1 \times DR(r) + w_2 \times DS(r) + w_3 \times CV(r) + \\ & + w_4 \times TWV(r) + w_5 \times D(r) + w_6 \times TWPP(r) + \\ & + w_7 \times DP(r) \end{aligned} \tag{3.1}$$

where:

$w_1, \ldots, w_7$	= weights of individual cost components
$DR(r)$	= duration of the route r
$DS(r)$	= distance of the route r
$CV(r)$	= number of violated capacity constraints
$TWV(r)$	= number of violated time windows constraints
$D(r)$	= sum of all delays for given time windows in route r (total delay)
$TWPP(r)$	= time window priority penalty*
$DP(r)$	= sum of delay penalties**

* In my problem I assume, that each point (pick up or delivery) can have multiple time windows with different priority thus the cost function has an additional variable $TWPP(r)$ time window priority penalty computed as a sum of *the number*

of time windows with higher priority divided by *number of time windows* for every point on the route.

** Delay penalty is computed for all points on the route r given a time window chosen for that point. The reason for this additional delay penalty variable is to introduce a non-linear penalty function that will force the algorithm to minimize the delay on the individual nodes rather than just minimizing the overall delay on the route. Based on the experiments with customers, we within the GoDeliver team conclude that it is better to have several smaller delays rather than just a few bigger delays. As a non-linear function, I have chosen the following one.

$$2 * exp(2 * (x + 1)) \tag{3.2}$$

The x variable in the non-linear penalty function stands for delay at the given point in hours. The largest penalty should be imposed on the time window violations, in order to direct the solution search towards more feasible routes. Based on the observations I have decided to use the following weights $w_1 = 0.005$, $w_2 = 0.005$, $w_3 = 0.3$, $w_4 = 0.2$, $w_5 = 0.8$, $w_6 = 0.001$ and $w_7 = 1$.

Algorithm 1: Route evaluation algorithm

```
   Input: route $r$
   Output: route cost
1  $n' \leftarrow nil$
2  for $n \in r$ do
3      if $n' \neq nil$ then
4          $DR(r) \leftarrow DR(r) + durationFromTo(n', n)$
5          $DS(r) \leftarrow DS(r) + distanceFromTo(n', n)$
           /* If a request has a multiple time windows select one with smallest
              time difference.                                               */
6          $timeWindow \leftarrow findBestTimeWindow(n, DR(r))$
7          $delay \leftarrow delayAtTW(timeWindow, DS(r))$
8          if $delay > 0$ then
9              $D(r) \leftarrow D(r) + delay$
10             $DP(r) \leftarrow DP(r) + 2 * exp(2 * ((delay/3600) + 1))$
11             if hardTimeWindow(timeWindow) then $TWV(r) \leftarrow TWV(r) + 1$
12         $a \leftarrow numOfTwWithHigherPriority(n, timeWindow)$
13         $b \leftarrow numberOfTimeWindows(n)$
14         $TWPP(r) \leftarrow TWPP(r) + a/b$
15     if isCapacityViolatedAtNode(n) then $CV(r) \leftarrow CV(r) + 1$
16     if isSkillViolatedAtNode(n) then $SV(r) \leftarrow SV(r) + 1$
17     $n' \leftarrow n$
   /* return the cost of the route $r$ computed using equation 3.1          */
18 return cost(r)
```

3.3.1 Complexity analysis and implementation issues

The evaluation algorithm iterates over individual nodes in the route and computes all variables required for the cost function 3.1. The main loop thus takes $\mathcal{O}(n)$, where n is the number of nodes in the route r. Computing distance, duration, and delay at the node n are done in constant time. Finding the best time window must iterate over all time windows in a given node but the number of time windows at a given node is negligible so I will assume that it takes a constant time. The same applies when determining if the capacity or the skill constraint is violated. Assuming this, the asymptotic complexity of Alg. 1 is $\mathcal{O}(n)$.

The main issue in the implementation of Alg. 1 is how to define and choose the best time window. If the node has only a single time window then we can return it, but in the case when a node has several time windows then, there are two possible outcomes.

- In the first case, the time at which the vehicle arrives at the node n is within one of the node's time windows. This is the best possible scenario that can happen because that time window can be returned as the best possible one.
- In the second case, none of the time window ranges covers a time in which a vehicle arrived at the node n. For this case, I used a simple approach of finding a time window (TW) immediately to the vehicle's arrival time in terms of time difference. As can be seen in Fig 3.2. If a distance to TW 1 is smaller than the distance to TW 2, first time window is selected otherwise the second is selected.

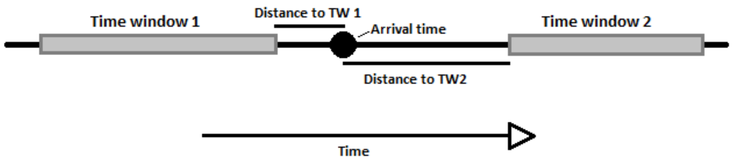


FIGURE 3.2: Time window selection

3.3.2 (Sub)route construction and selection

An important part of the routing algorithm is the part where we generate a new candidate (Alg. 2). Candidate is defined as a new (sub)route obtained by adding new request x to the previous route r. The candidate carries the information about the newly constructed (sub)route, vehicle, request, and route's cost given by the cost function 3.1.

Algorithm 3 describes the process of selecting a new candidate from the list of candidates based on the α parameter. The value of $\alpha \in \langle 0, 1 \rangle$, that plays an important role in the selection process, will be further explained in section 4.3.1. The candidate selected by this algorithm is used within the construction heuristics to construct part of the final solution.

The following algorithms will be used in the construction process of different construction heuristics (see next section).

Algorithm 2: Candidate generation

```
    Input: list of candidates $CL$
    Input: request $x$
    Input: vehicle $v$
    Input: route $r$
    Output: $CL$
1 $r' \leftarrow r \cup \{x\}$                          // insert x to route r
2 $r'' \leftarrow HillClimbing(r')$           // call HC algorithm 4 to improve r'
3 $CL \leftarrow CL \cup \{newCandidate(r'', v, x)\}$  // create and add new candidate to CL

4 return $CL$
```

Algorithm 3: Candidate selection and assignment

```
    Input: value $\alpha$
    Input: list of candidates $CL$
    Input: route $r$
    Input: list of unassigned requests $X$
    Output: candidate/nil, new route $r$, new list of unassigned requests $X$
1 $RCL \leftarrow filterUnfeasible(CL)$         // create restricted candidate list
    containing only feasible solutions
2 if $len(RCL) == 0$ then return $nil, r, X$
3 $sort(RCL, descending)$         // sort RCL based on cost value given eq.3.1
4 $maxIndex \leftarrow (1 - \alpha) \times (len(RCL) - 1)$                    // $\alpha \in \langle 0, 1 \rangle$
5 $selectedCandidate \leftarrow RCL[randomInt(0, maxIndex + 1)]$
6 $r \leftarrow selectedCandidate.route$  // update route r based on selectedCandidate
7 $X \leftarrow X \setminus \{selectedCandidate.x\}$          // remove request from X based on
    selectedCandidate
8 return $selectedCandidate, r, X$
```

3.4 Routing algorithm

The crucial part of the construction process is the routing algorithm that will generate a feasible solution based on the problem's constraints. The implemented routing algorithm is based on [25]. This algorithm is based on iterative improvements of individual routes and is embedded in the construction algorithm that could be either sequential or parallel. Compared to the other insertion algorithms, this algorithm does not try to find the best insertion position but rather accepts any feasible insertion position. This results in less complex calculations and decisions related to the problem specification.

The solution is represented as a permutation of pickup and delivery pairs (requests) rather than a one-dimensional permutation of all different locations. Meaning that the same identifier is assigned to the both pickup and delivery location. Since this representation relies on a simple decoder that the first occurrence of the identifier in the route is always pickup and the second occurrence is always delivery, both coupling and precedence constraints are handled and we no longer have to ensure both pickup and delivery points are assigned to the same vehicle and pickup must be visited before delivery. On the other hand, this representation may induce a redundancy in the locations, especially for the pickup points where a single pickup point might be assigned to the multiple delivery points. Capacity and time window constraints might be violated in this representation but both constraints are penalized by the cost function (see equation 3.1).

For the purpose of the VRPPDTW, a simple Hill climbing (HC) algorithm was selected as a route-improving algorithm. This algorithm tries to gradually improve a current route by swapping positions of points in the route until no further improvements are possible. The reason this algorithm was chosen is its simplicity and effectiveness.

Algorithm 4: Hill Climb routing algorithm v1

```
   Data: initial route $r$
   Result: improved route $r$
1  initialization;
2  while route was improved do
3      Evaluate($r$);
4      for each possible pair of locations $i, j$ in $r$ do
5          if $j$-th location is more urgent than $i$-th location then
6              Swap $i$ with $j$ to get new route $r'$;
7              Evaluate($r'$);
               /* cost of route $r$ is computed using equation 3.1 */
8              if $cost(r') < cost(r)$ then $r \leftarrow r'$ ;
9          end
10     end
11 end
12 return $r$
```

The first version of HC algorithm 4 swaps points *i* and *j* (*line 5*) only if the end of the *j*-th point time window ends before the end of the *i*-th point time window. The second version of this algorithm ignores the if condition at *line 5* and swaps every possible combination of points. In other words, the second version (algorithm 5) is being fully-greedy as it tries to explore the whole neighbor space.

Algorithm 5: Hill Climb routing algorithm v2

```
   Data: initial route $r$
   Result: improved route $r$
1  initialization;
2  while route was improved do
3      Evaluate($r$);
4      for each possible pair of locations $i, j$ in $r$ do
5          Swap $i$ with $j$ to get new route $r'$;
6          Evaluate($r'$);
           /* cost of route $r$ is computed using equation 3.1 */
7          if $cost(r') < cost(r)$ then $r \leftarrow r'$ ;
8      end
9  end
10 return $r$
```

3.5 The sequential construction algorithm

As mentioned before, the sequential construction heuristic algorithm (SQA) builds the routes one by one by selecting a request (pickup and delivery pair) and adding them at the end of the route. Then the Hill-Climbing algorithm is called to improve the current route. If the HC algorithm finds a feasible route then the algorithm selects another request. However, if the request cannot be inserted, HC returns an infeasible route, then the request is removed from the current route and inserted at the end of the next route. So this algorithm heavily relies on the HC algorithm to improve the quality of the route without the need to calculate the cost of every possible insertion position. Algorithm 6 describes the sequential construction procedure.

Algorithm 6: Sequential route constuction (SEQ)

```
Input: list of requests (PD pairs) X
Output: list of routes for each vehicle
1  $R \leftarrow []$                                  // array of initial empty routes
2  for each vehicle $v \in V$ do
3      while true do
4          $CL \leftarrow []$   // candidate list.  list of sub-routes and latest request
            assigned to that sub-route
5          for each unassigned request $x \in X$ do
               /* call Algorithm 2                                          */
6              $CL \leftarrow createAndAddCandidateToCandidateList(CL, x, v, R[v])$
           /* call Algorithm 3                                              */
7          $candidate, r, X \leftarrow selectCandidateAndUpdateRequests(CL, R[v], X)$
8          if $candidate = nil$ then break
9      $R[v] \leftarrow r$
   /* if there are some requests not assigned make the last route unfeasible by
      assigning all points to it                                            */
10 if $len(X) > 0$ then
11     $r \leftarrow R[|V|]$                              // last vehicle's route
12     for each unassignedRequest $x \in X$ do
13         $r \leftarrow r \cup \{x\}$          // insert $x$ to route of the last vehicle
14     $r' \leftarrow HillClimbing(r)$          // call HC algorithm 4 to improve $r$
15     $R[|V|] \leftarrow r$
16 return $R$
```

3.6 The parallel construction algorithm

The parallel construction algorithm builds all routes simultaneously. As mentioned previously an initial estimate of the number of vehicles is required. To estimate an initial number of vehicles, the resulting number of routes from Solomon's sequential construction [7] can be used. Traditional parallel construction algorithms insert the selected request to the best possible route. If the request cannot be successfully inserted a new route is added.

3.6.1 Best route parallel construction

The construction process of the algorithm 7 works as follows: In each iteration of the main for loop, the selected request is inserted in the route in which the insertion produces a new route with the lowest cost among all routes. Thus the algorithm tries to construct the routes with similar costs.

Algorithm 7: Parallel best route constuction (PBR)

```
Input: list of requests (PD pairs) X
Output: list of routes for each vehicle
1  $R \leftarrow []$                                  // array of initial empty routes
2  for each unassigned request $x \in X$ do
3      $CL \leftarrow []$        // candidate list.  list of sub-routes and latest request
         assigned to that sub-route
4      for each vehicle $v \in V$ do
           /* call Algorithm 2                                                    */
5          $CL \leftarrow createAndAddCandidateToCandidateList(CL, x, v, R[v])$
       /* call Algorithm 3                                                        */
6      $candidate, r, X \leftarrow selectCandidateAndUpdateRoute(CL, [], X)$
       /* if no feasible candidate exist assign first candidate...                */
7      if $candidate = nil$ then
8          $candidate \leftarrow CL[0]$
9          $X \leftarrow X \setminus \{candidate.x\}$  // remove request from $X$ based on $candidate$
10     $R[candidate.v] \leftarrow candidate.r$
11 return $R$
```

3.6.2 Best request parallel construction

The second parallel construction heuristics (Alg. 8) do not only try to find the best route for each request but also find the best request for each route. The best-unrouted request is the one whose insertion causes the smallest increase in the cost of the partial solution.

Algorithm 8: Parallel best request constuction (PBQ)

```
Input: list of requests (PD pairs) X
Output: list of routes for each vehicle
1  $R \leftarrow []$                                  // array of initial empty routes
2  while $len(X) > 0$ do
3      $CL \leftarrow []$        // candidate list. list of sub-routes and latest request
        assigned to that sub-route
4      for each unassigned request $x \in X$ do
5          $RBCL \leftarrow []$     // route best candidate list. list of sub-routes and
            latest request assigned to that sub-route
6          for each vehicle $v \in V$ do
               /* call Algorithm 2                                              */
7              $RBCL \leftarrow$
                 $createAndAddCandidateToCandidateList(RBCL, x, v, R[v])$
8          $CL \leftarrow CL \cup selectBestCandidateFrom(RBCL)$
       /* call Algorithm 3                                                      */
9      $candidate, r, X \leftarrow selectCandidateAndUpdateRoute(CL, R[v], X)$
       /* if no feasible candidate exist assign first candidate...              */
10     if $candidate = nil$ then
11         $candidate \leftarrow CL[0]$
12         $X \leftarrow X \setminus \{candidate.x\}$  // remove request from X based on candidate
13     $R[candidate.v] \leftarrow candidate.r$
14 return $R$
```

3.6.3 Parallel insertion k-Regret construction heuristic

Similar to the previous construction heuristic, this one computes the best insertion value of each request into each route and inserts the one with the highest k-Regret value. K-Regret value is calculated within the insertion procedure as $\sum_{m=1}^{M}(f(m,i) - f(i,i))$ (function *computeKRegret* at *line 5* of Alg.9), where $f(m,i)$ is cost of i-th request inserted in m-th best route and represents the regret of not inserting the current request into the current route. In other words, the regret value is a measure of the potential cost that could be paid if a given request were not immediately inserted. This feature is particularly useful for highly constrained problems, as it drives the algorithm towards the search for feasible solutions. The main focus is to limit as much as possible the myopic behavior of the classical insertion procedures, which for mixed scheduling and routing problems with additional constraints is particularly harmful [17].

Algorithm 9: Parallel insertion k-Regret (PIR)

```
Input: list of requests (PD pairs) X
Input: list of vehicles V
Input: k-regret value k
Output: list of routes for each vehicle
1  $R \leftarrow []$                                    // array of initial empty routes
2  while $len(X) > 0$ do
3      $CL \leftarrow []$        // candidate list.  list of sub-routes and latest request
        assigned to that sub-route
4      for each unassigned request $x \in X$ do
5          $CL \leftarrow CL \cup computeKRegret(k, x, R, V)$
       /* call Algorithm 3                                                   */
6      $candidate, r, X \leftarrow selectCandidateAndUpdateRoute(CL, [], X)$
       /* if no feasible candidate exist assign first candidate...           */
7      if $candidate = nil$ then
8          $candidate \leftarrow CL[0]$
9          $X \leftarrow X \setminus \{candidate.x\}$   // remove request from X based on candidate
10     $R[candidate.v] \leftarrow candidate.r$
11 return $R$
```

Chapter 4

GRASP implementation

This chapter presents the basic overview of metaheuristics. We begin with an introduction of simple metaheuristics called random and semi-greedy multi-start procedures, and describe how these procedures differ. Then we introduce a GRASP metaheuristic and describe in detail the implementation of GRASP for VRP.

4.1 Metaheuristic

According to [26] a metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or techniques for developing heuristic optimization algorithms. Notable examples of metaheuristics include genetic/ evolutionary algorithms, tabu search, simulated annealing, variable neighborhood search, (adaptive) large neighborhood search, and ant colony optimization, among many more.

4.2 Random multi-start procedure

A randomized multi-start technique is the most basic type of metaheuristic. This approach is illustrated in Algorithm 10. This method creates a random solution until the stop condition is fulfilled, at which point it outputs the best solution discovered. In a *line 2*, a solution is formed by adding a new ground element from the set of elements to partial-solution. This new ground element is picked at random from the candidate set of ground elements. elements.[31]

Algorithm 10: Pseudo-code of randomized multi-start procedure

```
   Result: feasible solution $r$
1  while stop condition do
2      r′ ← randomSolution();
3      if r′ is not feasible then
4          repairSolution(r′)
5      end
6      if $cost(r') < cost(r)$ then
7          r ← r′
8      end
9  end
10 return r;
```

4.3 Semi-greedy multi-start procedure

A semi-greedy multi-start method is yet another type of metaheuristic. The pseudo-code for this process is available in Algorithm 11. The sole difference between this procedure and Algorithm10 is a semi-greedy building step (*line 2*). A semi-greedy solution is formed by adding a ground element from the restricted candidate list (RCL) of ground elements. In a minimization problem, an RCL is a list of ground elements regulated by the parameter $\alpha \in \langle 0, 1 \rangle$, where $\alpha = 1$ leads to a pure-greedy solution because only the best ground element is placed in the RCL and $\alpha = 0$ leads to a pure-random solution because all ground elements are placed in the RCL. The RCL is then used to generate a new element at random. [31]

Algorithm 11: Pseudo-code of randomized multi-start procedure

```
Result: feasible solution r
while stop condition do
    r′ ← semiGreedySolution();
    if r′ is not feasible then
        repairSolution(r′)
    end
    if $cost(r') < cost(r)$ then
        r ← r′
    end
end
return r;
```

4.3.1 Effect of alpha value on semi-greedy construction

Figure 4.1 shows the distribution of the solution values after each iteration of the semi-greedy construction procedure. For different α values, a total of 1000 solutions values were constructed by the PBR construction heuristic (Algorithm 7). All solutions were constructed on VRP instances with 100 customers and 10 vehicles. The distribution shows that the solutions are less spread for the higher α values (> 0.6) compare to the lover α values (< 0.6). This also means that semi-greedy procedure is less likely to escape local optimum with a high α value - as can be seen with a greedy approach (with $\alpha = 1$), the best-produced solution has a cost *12 700* compare to the best solution produced by semi-greedy procedure with $\alpha = 0.8$, with a cost *12 450.* This is illustrated better in a figure 4.2. In this example, even a semi-greedy construction procedure with $\alpha = 0.9$ was not able to escape a local optimum thus not improving. I have decided to select two promising alpha values (**0.6** and **0.7**) for future experiments because these values produced solutions with the best quality

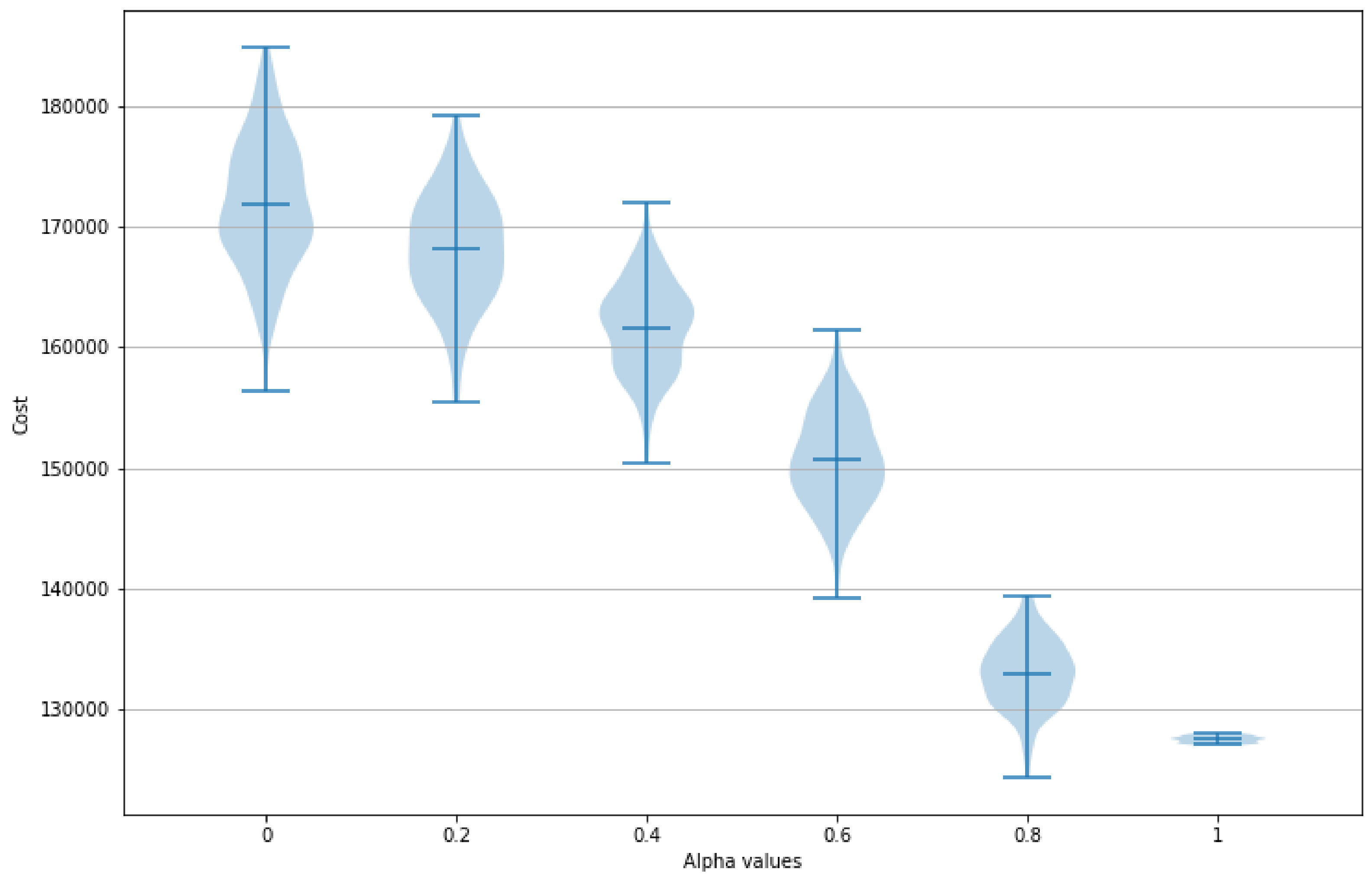


FIGURE 4.1: Distribution of the solution cost obtained by the semi-greedy construction procedure as a function of α value. (1000 repetitions of a minimization problem of VRP on instances with 100 customers and 10 vehicles)

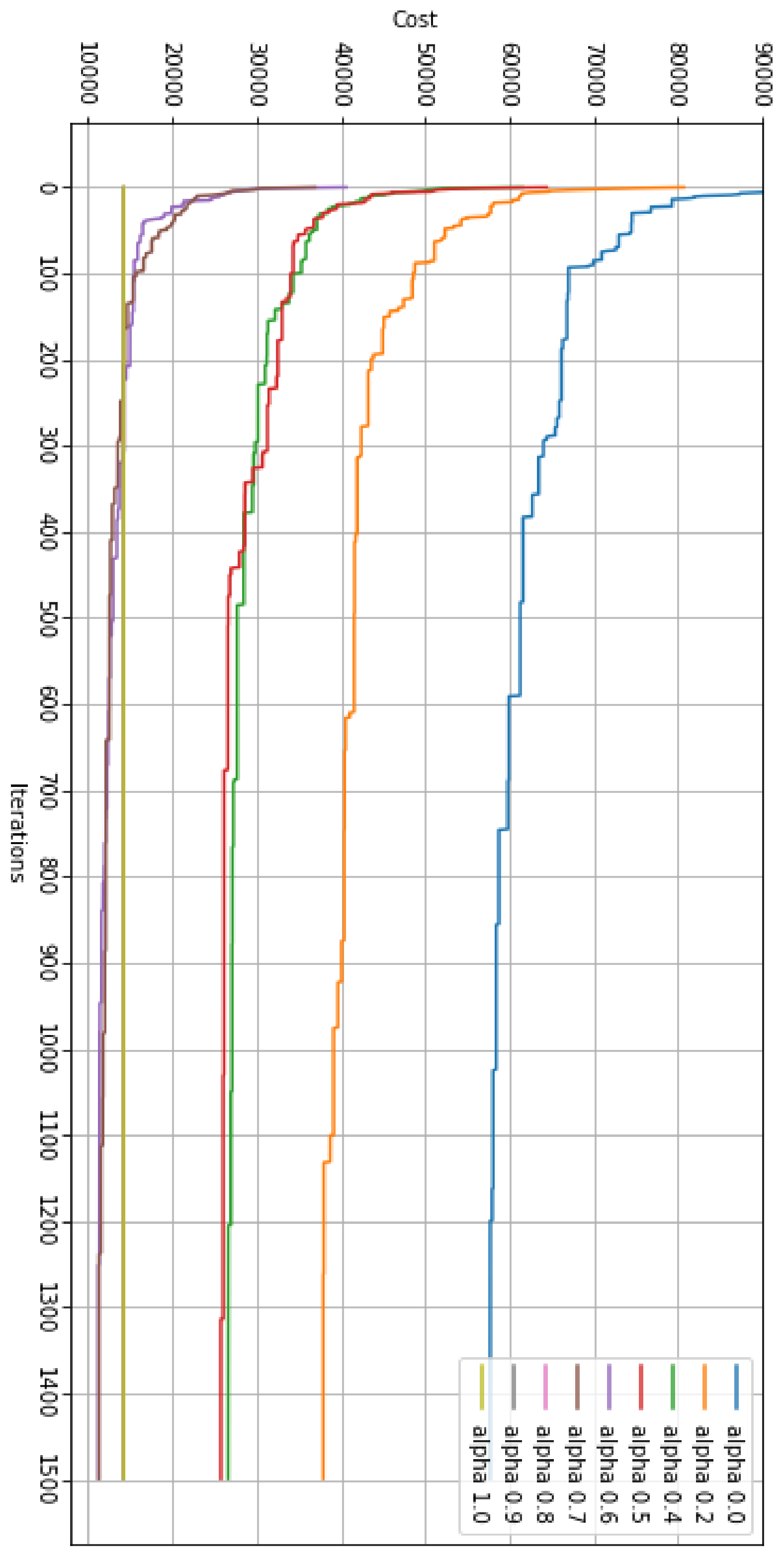


FIGURE 4.2: Average cost of best solution obtained by construction heuristic with different alpha values

4.3.2 Parallel vs sequential multi-start procedure

In a sequential multi-start procedure, we build a solution one by one, keeping the best solution we found so far. Since we are not dependent on the best solution we found so far, we can easily use multiple processors to find the solution in parallel. Each processor can construct the solution by itself and then compare it with the best solution we found. The construction of the solution takes way more time than comparing the solution thus the CPU overhead is minimal as can be seen in figure 4.3. For testing purposes, a used *4 core* CPU with no multi-threading and the results show more than **3x** speedup in terms of execution time after 1000 iterations, for a Multi-start procedure with PBR (Alg. 7) semi-greedy construction procedure.

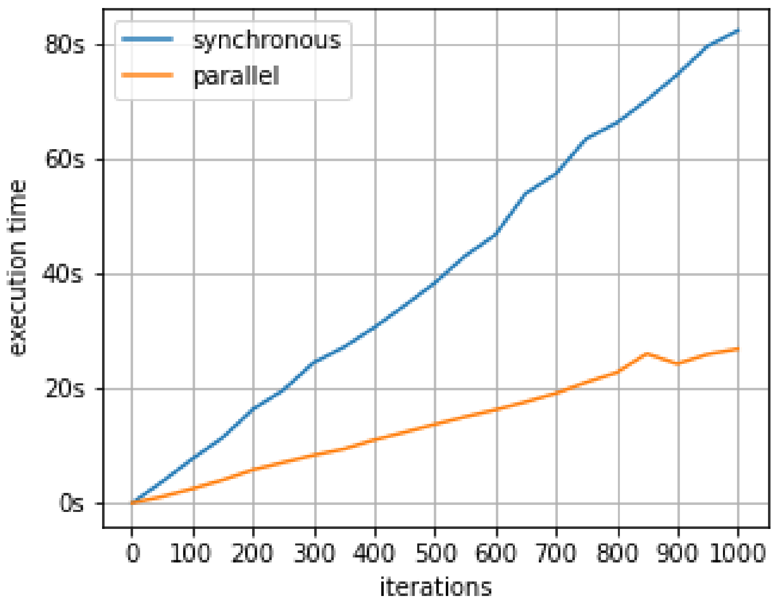


FIGURE 4.3: Effect of parallelization of the GRASP algorithm with PBR 7 semi-greedy construction procedure

4.4 GRASP metaheuristic

The greedy randomized adaptive search procedure (GRASP) is a multi-start metaheuristic that enjoys wide success in practice, with an extraordinarily broad range of applications to real-world optimization problems [24]. Also, it is among the most effective metaheuristics for solving combinatorial optimization problems [31]. GRASP is a hybridization of a semi-greedy multi-start procedure with a local search algorithm 12. The procedure works in two phases - construction and improvement. In the first phase, a construction heuristic builds a feasible solution. In the second phase, its neighborhood is investigated by a local search procedure, until a local minimum is found. The best overall solution is kept as a result.

Algorithm 12 illustrates a simple GRASP metaheuristic for minimization problem. After initialization in *line 1*, the main GRASP iterations are carried out in the while loop in lines *2 to 11*. At first a semi-greedy solution r' is constructed in *line 3*. Sometimes a semi-greedy solution might not be feasible, thus a repair procedure must be invoked in *line 5* to make a necessary changes to r' so that it becomes feasible. Alternatively an infeasible solution r' might be discarded and followed by another iteration until a feasible solution is constructed, but in this case a stop condition in *line 2* should take this into account. In *line 7*, local search algorithm is applied. We used an Hill-climbing algorithm (Algorithm 4) described in chapter 3.4. If the objective function $cost(r')$ (function 3.1 from chapter 3.3) is better than objective function $cost(r)$ then a new local minimum is saved in *line 9*. The best solution found during the main GRASP loop is then returned in *line 12* as a GRASP solution.

Algorithm 12: Pseudo-code of GRASP

```
   Result: feasible solution r
1  r ← null
2  while stop condition do
3      r′ ← semiGreedySolution();
4      if r′ is not feasible then
5          repairSolution(r′);
6      end
7      r′ ← localSearch(r′);
8      if $cost(r') < cost(r)$ then
9          r ← r′;
10     end
11 end
12 return r;
```

The biggest drawback of most metaheuristics, including GRASP, is the absence of effective stopping criteria. Commonly used stopping criteria are the maximum number of iterations or the maximum number of consecutive iterations without improvements or stabilization of the elite set of solutions found during the search. Sometimes the metaheuristic algorithm can quickly find the best solution (as often happens in GRASP [31]) thus performing an exaggerated and unnecessary number of iterations.

4.5 GRASP algorithm for VRPPDTW

4.5.1 Recap on problem formulation

Let $G = (V, A)$ be a complete digraph with set of edges (arcs) $A = \{\{i, j\} : V \times V, i \neq j\}$. Each node $i \in V$ has a time window $\langle t_i, t_j \rangle$ and a duration d_{ij} associated with edge $(i, j) \in A$. The route (path) in the graph G consist of pickup nodes $P = \{1, \ldots, n\}$ and corresponding delivery nodes $D = \{n + 1, \ldots, 2n\}$. Union of P and D creates an original set of vertices in G. It is possible that different pickup/drop nodes may represent the same geographical location. VRPPDTW problem consist of finding the optimal set of routes in the transportation network (graph G) for a fleet of vehicles [25].

4.5.2 GRASP construction

The construction of a new solution is done by a sequential algorithm described in chapter 3.5 or by one of the parallel algorithms described in chapter 3.6. Each iteration of a construction phase starts by picking up the best request from the set of unassigned requests and inserting it into the best position in the given route (in case of sequential construction algorithm 6) or inserting it into the best route of a given set of routes (in case of parallel construction algorithms 7, 8, 9). The design of these algorithms always produces a feasible solution or no solution at all, thus no repair procedure is needed.

4.5.3 Local search

The local search phase seeks to improve each solution built during the construction phase by relocating the requests in-between routes. From the nature of the construction heuristics, where during each iteration a hill-climbing algorithm (Algorithm 4) is called to insert a request to the best position in the route and to eventually reorder the rest of the route (improving partial solution as a whole), it is not necessary to try to further improve the individual routes without moving at least one request into a different route. The procedure of the Local search phase is shown in Algorithm 13.

Algorithm 13: Local Search algorithm for GRASP metaheuristic

```
Input: solution M
Output: improved solution M′
/* sorted routes, in descending order, based on their objective (cost) value */
1  $R \leftarrow getSortedRoutes(M)$
2  $improved \leftarrow true$
3  while $improved$ do
4      $improved \leftarrow false$
5      outer:
6      for each route $r \in R$ do
           /* find request that, by removing, decrease route r objective (cost) value most */
7          $newR1 \leftarrow nil$
8          $bestRequest \leftarrow -1$
9          $bestCostDecrease \leftarrow 0.0$
10         for each request $i \in r$ do
               /* Remove request x from the route r */
11             $r' \leftarrow r \setminus \{x\}$
12             $hillClimb(r')$
13             $costDecrease \leftarrow cost(r) - cost(r')$
14             if $bestRequest == -1$ or $bestCostDecrease < costDecrease$ then
15                 $newR1 \leftarrow r'$
16                 $bestRequest \leftarrow x$
17                 $bestCostDecrease \leftarrow costDecrease$
           /* Find new route to insert 'bestRequest' */
18         for each route $r2 \in R$ do
19             if $r == r2$ then continue
20
               /* Add bestRequest to route textitr2 */
21             $newR2 \leftarrow r2 \cup \{bestRequest\}$ $hillClimb(newR2)$
               /* Check if cost of new routes is better than cost of old routes */
22             if $feasible(newR2) == feasible(r2)$ then
23                 if $cost(newR2) + cost(newR1) < cost(r) + cost(r2)$ then
24                     $improved \leftarrow true$
25                     $r \leftarrow newR1$
26                     $r2 \leftarrow newR2$
27                     break outer
28 $S' \leftarrow constructSolution(R)$
29 return S′
```

The LS algorithm accepts a solution and eventually returns an improved solution. We start by sorting all routes based on their cost value in the descending order (*line 1*). Then we try to move request x from some route $r \in R$ to a different route $r' \in R$. The whole procedure is divided into two for-loops. In the first loop (lines *10 to 17*) we find a request x that has the greatest effect on the route's cost. Meaning, we are looking for a request, whose removal from the route r gives us the best improvement in terms of the solution's cost. Then in the second loop (lines *18 to*

27) *bestRequest* is inserted to a different route *r*2 if two conditions are met. Newly constructed route (with request *bestRequest*) $r2 \cup \{bestRequest\}$ does not increase the number of unfeasible routes (the route is still feasible or unfeasible if the old route was also unfeasible). And the new solution has an overall cost lower than the previous solution (line *24)*. We continue with this procedure until there are no more requests that can be relocated (while loop - lines *3 to 27*).

4.5.4 Effect of LS algorithm on solution's quality

My next experiment was conducted by analyzing the performance of this LS algorithm 13 with the following conclusion. Solutions constructed by PBR (Algorithm 7) and PBQ (Algorithm 8) with a HillClimb procedure (Algorithm 4) on an instance with 100 requests and 50 vehicles were improved on average only by **3.3%** and **3%** respectively. As can be seen in figures 4.4 and 4.5 and 4.6. The fact[1] that the original solution (plan) is optimal within the individual routes in the plan is causing this low improvement of the new solution (plan) obtained after applying the LS algorithm. Thus there is little to no space to further improve the plan by moving the requests to different routes. Moving a request from route *A* to route *B* improves the cost of route *A* but makes the cost of route *B* worst. And the cost of the solution is likely to be higher than the cost of the original solution.

With a time restriction we have within our GoDeliver's planning algorithm, it might not be worth spending extra time trying to improve the solution with this LS algorithm. On the other hand, various weights in the cost function 3.1 may improve the difference in cost of the solutions. Also if the time restriction shows not to be an issue I don't see any drawbacks of using this LS algorithm.

[1]HC algorithm 4 in the construction phase always finds the best permutation of requests for a given route

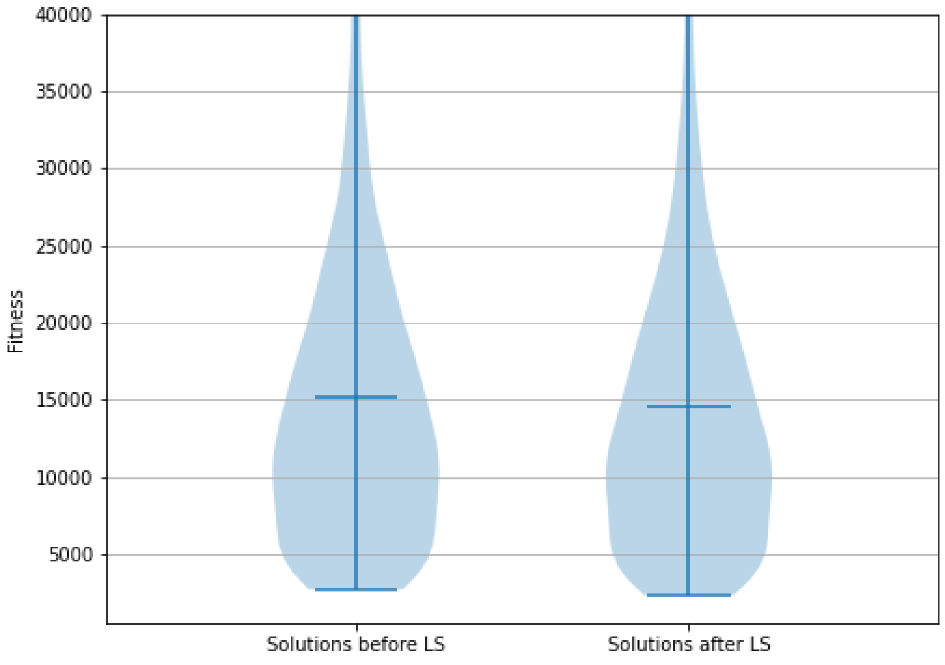


FIGURE 4.4: Distribution of solutions for PBR construction heuristic

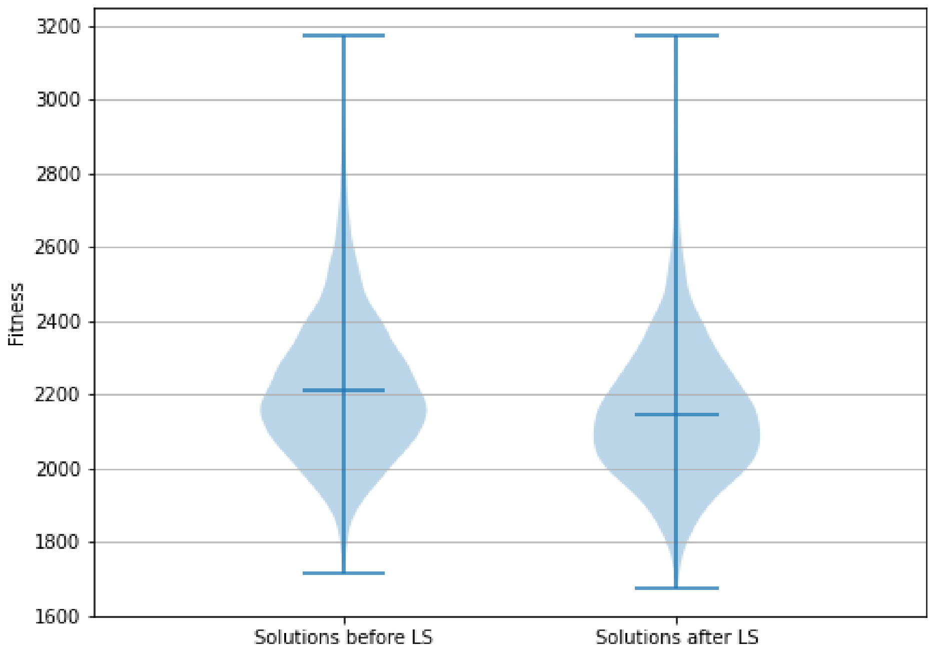


FIGURE 4.5: Distribution of solutions for PBQ construction heuristic

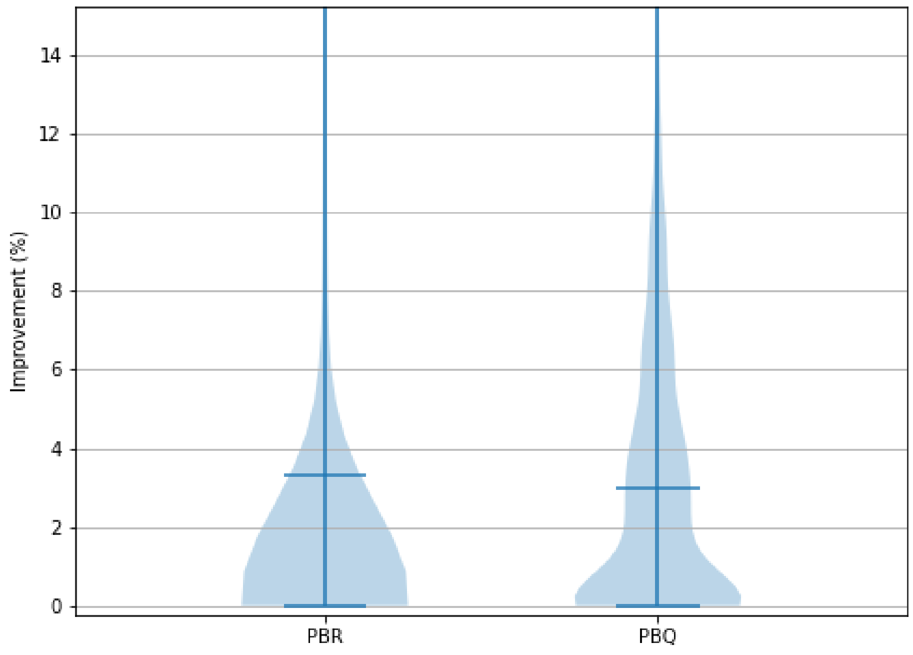


FIGURE 4.6: Distribution of solution's cost difference before and after application of LS procedure

Chapter 5

Computational Experimentation

5.1 Data-set characteristics

The algorithms have been tested using several data sets of real-world requests (pickup and delivery) with time windows from a Prague-based restaurant. The data set includes 50-300 real locations, and the time windows range from 10:00 to 15:00. Resulting on average in 10-30 customers being served in the same time frame. Time windows in these data sets were both hard (opening and closing times of the branch) and soft (time frame specified by the customer). Additionally, two types of vehicles with varying cargo capacities are included in the data sets - cars and bikes/scooters. Vehicles of any type could visit any customer since no special skills were required. In the section 3.6, it is noted that the parallel version of the construction heuristics algorithm requires an initial estimate of how many vehicles are involved. Data sets used to test the algorithm included the maximum fleet size and the initial location for all vehicles, which were used in parallel heuristic algorithms.

5.2 Existing solvers and baseline algorithms

As baselines, I used several existing solvers/construction heuristics. The first baselines are a simple insertion heuristic and HALNS solver implemented in [36]. The third baseline is the OR-Tools planner[1] that is a part of the GoDeliver planning algorithm. The fourth and the last is a so-called "Jackpot solver"—a semi-greedy multistart procedure (described in section 4.3) with insertion heuristic as a construction procedure implemented based on research by my co-worker David Mokoš.

5.3 Comparison criteria

Evaluation datasets used in measurements are real-world instances of different problem sizes (number of requests). These datasets were created for testing purposes within a GoDeliver's system and are based on the real data from a selected company, which delivers food from several restaurants in Prague and in total contains more than 10000 requests between 1st of January 2020 and 10th of July 2020. These datasets are described in detail in [36].

For instances with more than a few tens of requests, an optimal solution is nearly impossible to obtain (meaning that the optimal cost value is unknown). Furthermore, the cost of the solution given by the fitness function 3.1 does not provide sufficient information about the quality of the solution. This information was lost

[1]ILP/MILP solver developed by Google. https://developers.google.com/optimization

when I simplified the search procedure by transforming the multi-objective problem to a single-objective problem. Therefore, it is necessary to use another metrics (transform the cost to original indicators). In my comparison, I will use the following indicators:[2]

- **delivery delay per request** - Request's delay time given by its time window. Difference between delivery time and request's time window upper bound.
- **delivery time per request** - Time an order spent in the vehicle. Difference between delivery time and pickup time.
- **total delivery time** - Sum of all delivery times and waiting times for all vehicles.
- **total distance traveled** - Sum of total distance traveled by all vehicles.
- **execution time** - How fast the solution was found. This metric is not relevant to the solution's quality but is an important deciding factor since I assume a limited time constraint for a solver.

From these values, I have deduced the quality of the solution. Where lower values indicate a better solution. These metrics have also proven to be more user-friendly than a value for the cost function, as I've discovered from my research. The delay is more significant than the duration or distance in the VRP with time windows. In all cases, the maximum fleet size is limited but the time windows are not fixed; this is due to characteristics of those data sets. Most research assumes an unlimited number of vehicles, but in the real world, most businesses have a limited number of vehicles or people.

For the experiments with different construction heuristics (algorithms 6, 7, 8 and 9) for a GRASP solver I used a cost value produced by the cost function 3.1.

5.4 Comparing the construction heuristics

Three of all four implemented heuristics (SEQ, PBR, PBQ) are based on the heuristics described in Manar I. Hosny's [25] thesis. So I expected the results of these functions to be very similar compared to the original counterparts. The sequential construction (SEQ) heuristic has a great usage if your goal is also to minimize the fleet size or you don't know the minimum size of your fleet. As none of these use-cases applied to my data sets I have decided not to include the results of the SEQ heuristic. Another reason not to include the SEQ heuristic in the results is that the requests are assigned to the first vehicle until the route is feasible. So if all requests have soft time windows or their time windows are sufficiently big, the solution will contain all requests assigned to the first vehicle only. From the feasibility point of view, the final plan is feasible but not optimal as only the first vehicle is doing all work.

5.4.1 Implementation issues and complexity analysis

Based on the measurements shown in the table 5.1 - the Parallel Best Route (Alg. 7) (PBR) heuristic shows slightly better overall results compared to the Parallel Best Request (Alg. 8) (PBQ) heuristic. One of the reasons for this behavior might be caused

[2]Another reason is, that these are the metrics that my coworker used in his thesis [36]. So I can easily compare the GRASP performance with the existing solver used in GoDeliver's planning algorithm.

Instance (requests-vehicles)	**PBQ**				
	delay per order(min) (avg/max/median)	duration per order (min) (avg/max/median)	total distance traveled (km)	total time spent (min)	time (ms)
150-35	0.4/13.1/0	19.8/60.5/18.4	734	3029	5997
150-20	1.8/54.2/0	20.3/59/18.7	802	3201	14091
150-15	4.2/82.6/0	23.2/71.2/19.6	811	3142	18125
150-10	43.6/138.2/32	22/129.1/15.7	1020	3522	42560
100-8	19.4/125.8/0	27.2/116/19.3	648	2466	7184
70-6	25.7/143.6/0	25/117.3/18	531	1739	2156
50-5	20.3/99.6/0	25.6/105.4/17.5	382	1403	1070
Instance (requests-vehicles)	**PIR**				
	delay per order(min) (avg/max/median)	duration per order (min) (avg/max/median)	total distance traveled (km)	total time spent (min)	time (ms)
150-35	0.6/29.5/0	19.2/64/16.6	730	3251	6010
150-20	1.4/29.4/0	19.7/55.2/17	812	2900	11900
150-15	23/142.5/0	17.3/75.7/15.2	1141	2685	17770
150-10	53.6/222/25.5	24.2/168/16.4	1271	4520	34800
100-8	51/175/33.2	23.7/146/16	883	1242	7180
70-6	49.8/176/27.8	17.6/92.6/15	573	1489	2140
50-5	32/160/1.2	16/57.9/13.2	415	1094	760
Instance (requests-vehicles)	**PBR**				
	delay per order(min) (avg/max/median)	duration per order (min) (avg/max/median)	total distance traveled (km)	total time spent (min)	time (ms)
150-35	0.5/17.5/0	20.4/63.4/16.9	1283	2887	149
150-20	1.2/28/0	20/56.3/16.7	1425	2774	263
150-15	4.7/57.9/0	21/88.9/16.6	1602	2589	460
150-10	22.7/125/0	26.3/104.8/19.8	1584	3562	890
100-8	12.5/73.2/0	21.6/66.4/16.6	1072	2054	137
70-6	39.7/124.4/40	18.4/45/16.6	787	1469	92
50-5	5.7/42.7/0	17.6/59/15	482	907	24

TABLE 5.1: Result of construction heuristics

(in the PBQ heuristic case) by inserting the best requests (based on their insertion cost value) earlier in the construction phase, leaving the worst requests to be inserted last. This process causes the beginning of the route to be optimized, but the end will include the customers that did not fit nicely earlier in the final route. This problem could be solved using something called k-Regret value [17] described in chapter 3.6.3. To my surprise, the Parallel insertion k-Regret construction heuristic (Alg. 9) (PIR) showed even worse results than the PBQ heuristic.

Results also show that increasing the number of vehicles significantly reduces the computation time in both cases. Increasing the number of vehicles reduces the computation time by more than 3 times. The reason for this is the number of times an HC algorithm must be called. Finding a solution with more but shorter routes is less computationally expensive than finding a solution for less but longer routes.

5.4.2 Comparing with an exiting insertion heuristic

Table 5.2 shows the results of the new PBR and PBQ heuristic and currently used insertion heuristic described in [36]. On average, solutions found by the insertion heuristics are slightly better in terms of delay per order and slightly worst based on the duration per order and time spent metrics. Solutions generated by both PBR and PBQ heuristics are similar in total time spent and outperform insertion heuristic by more than *25%* By tweaking the weights parameters of the cost function (Function 3.1, the quality of the solution produced by the PBR and PBQ heuristic could be improved to match the quality of the solutions generated by the insertion heuristic.

In summary, the insertion heuristic is better in delay per order metrics but worst in duration per order metric compares to the PBR and PBQ construction heuristics.

5.4.3 Heuristics as an initial solution for OR-Tools

In the next experiment, I compared the quality of solutions produced by the OR-tools solver with initial solutions provided by different construction heuristics. Table 5.3 shows the results of the experiment. As all measurements show, the quality of the solutions found by the OR-Tools solver is heavily optimized in delay per order and total distance traveled metrics. On the other hand in all cases, the OR-tools found a solution with a slightly worst duration per order and total time spent metric. As the difference in the last two metrics is minor, the OR-tools is still a great solver if given an initial solution.

Instance (requests-vehicles)	**Existing Insertion heuristic**				
	delay per order(min) (avg/max/median)	duration per order (min) (avg/max/median)	total distance traveled (km)	total time spent (min)	time (ms)
300-50	0.1/7.8/0	28.8/64.5/25.3	853	5456	359
150-35	0/6.2/0	24.6/64.5/20	636	4168	144
100-15	0.2/2.2/0	24/70.6/18.8	500	2701	85

Instance (requests-vehicles)	**PBR**				
	delay per order(min) (avg/max/median)	duration per order (min) (avg/max/median)	total distance traveled (km)	total time spent (min)	time (ms)
300-50	0.6/24.6/0	17.5/59.7/14.6	1756	3311	245
150-35	0.1/17.6/0	19.5/56.7/16.6	1321	2762	157
100-15	1.2/13.7/0	20.3/48.1/17.4	1038	1616	117

Instance (requests-vehicles)	**PBQ**				
	delay per order(min) (avg/max/median)	duration per order (min) (avg/max/median)	total distance traveled (km)	total time spent (min)	time (ms)
300-50	0.4/11.5/0	20/51.8/18.7	817	3846	9737
150-35	0.6/18.2/0	20/61.2/17.7	784	3019	5163
100-15	3.8/72.4/0	19.5/58.7/17.4	612	1952	2474

TABLE 5.2: Result of existing insertion heuristic with PBR and PBQ

Combination/ heuristic	delay per order(min) (avg/max/median)	duration per order (min) (avg/max/median)	total distance traveled (km)	total time spent (min)	time (s)
Insertion + Or-Tools	0.08/4.8/0	28.8/64.4/26.2	815	5446	228
Insertion only	0.08/7.6/0	28.8/64.5/25.3	853	5456	3
PBQ + Or-Tools	0.01/2/0	26.2/69.3/24.6	716	4956	154
PBQ only	0.2/13.2/0	21.8/63/18.4	880	4132	12
PBR + Or-Tools	0.02/1.8/0	24.6/71.3/22.2	1068	4650	141
PBR only	0.5/22/0	17.4/57/15.7	1668	3292	4
PIR + Or-Tools	0.06/4.5/0	23.3/67.8/20.3	749	4417	153
PIR only	0.4/22.5/0	20.9/67/16.8	884	3853	11

TABLE 5.3: Result of PBR, PBQ a PIR and insertion heuristics as initial solution to OR-tools solver on instances with 300 customers and 50 vehicles

5.5 Comparing the results of GRASP with different construction heuristics

Figure 5.1 shows the average cost values of 10 runs of PBR, PBQ, and PIR (algorithms 7, 8 and 9) procedures during a 1500 iteration search on instances with 100 customers and 10 vehicles. PBQ procedure produces the overall best solutions (in terms of cost). On the other hand, the solutions produced by the PIR procedure scored worst in terms of cost. Since both PBQ and PIR heuristics were similar in the execution time (their time complexity is the same) and the solutions produced by the PBQ procedure were more than 2× better, I decided not further investigate the performance of the GRASP solver with PIR heuristic

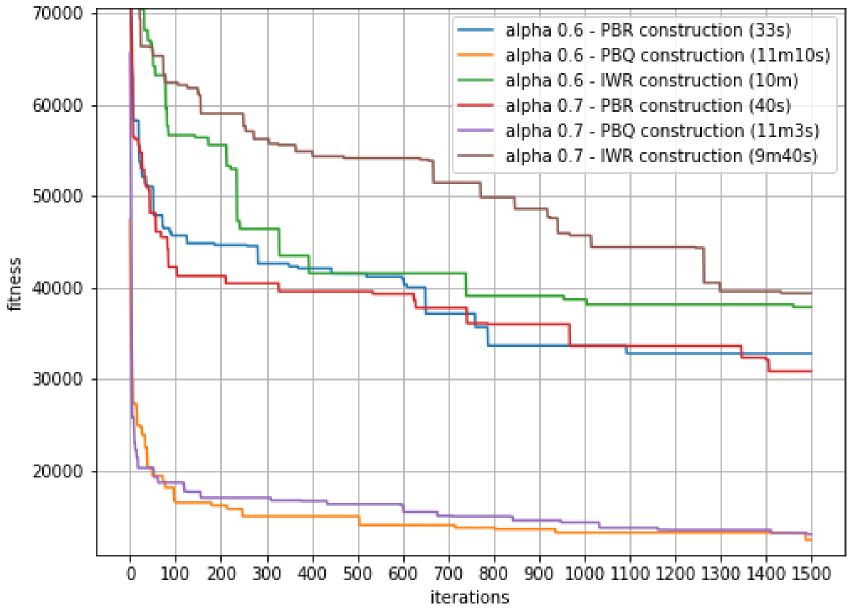


FIGURE 5.1: Cost evolution of the GRASP algorithm with different construction heuristic

Next, I measured the performance of the GRASP solver with construction heuristics using different Hill Climbing (HC) procedures (HCv1 - Algorithm 4 and HCv2 - Algorithm 5). As can be seen in figures 5.2, 5.3 and 5.4. Both HC procedures in the PBQ construction heuristic produced solutions similar in terms of cost, but the combination of HCv2 with PBQ heuristic required 25% less time to perform the search. On the other hand, the combination for the PBR construction heuristic the second version of the HC procedure (HCv2) produced solutions with better quality (on average by 25%), but the search took on average 5× more time.

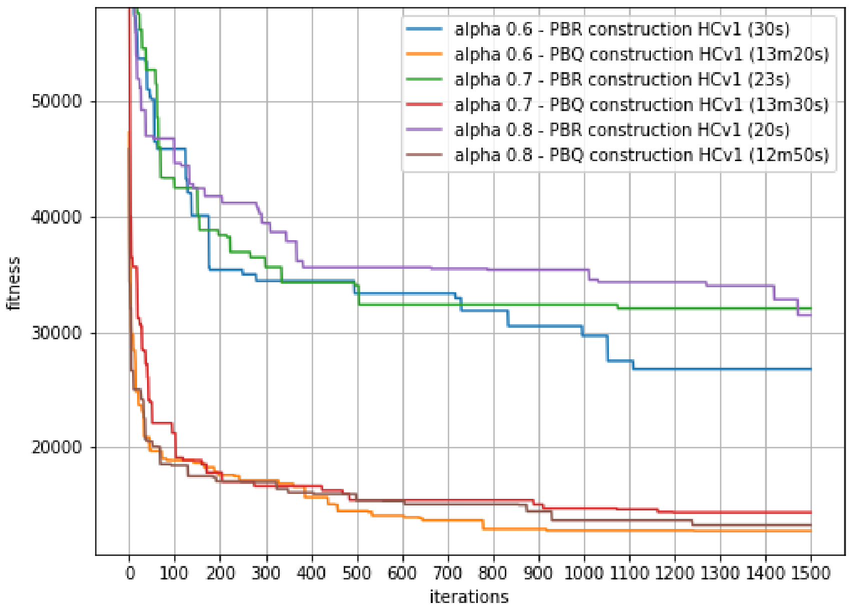


FIGURE 5.2: Cost of best solution found by GRASP with PBR and PBQ construction functions with less greedy HC algorithm 4

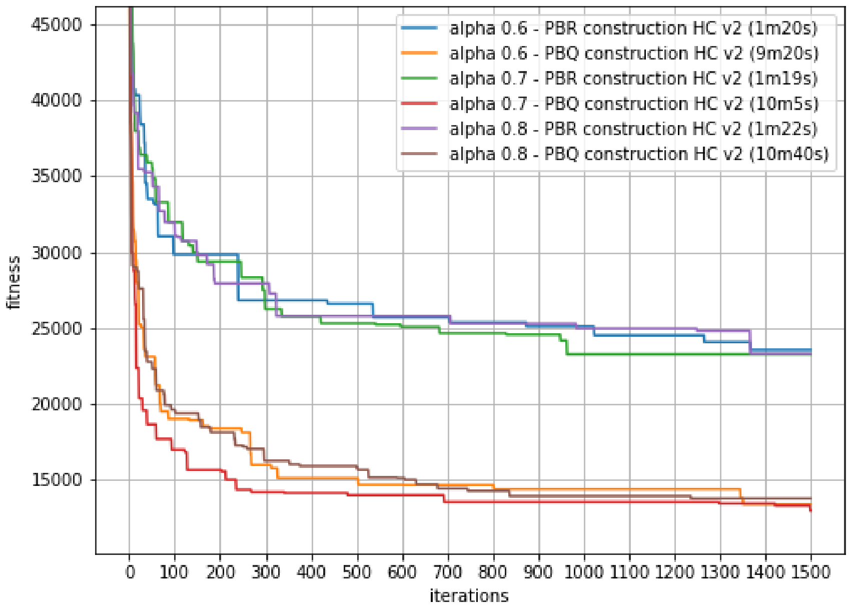


FIGURE 5.3: Cost of best solution found by of GRASP with PBR and PBQ construction functions with more greedy HC algorithm 5

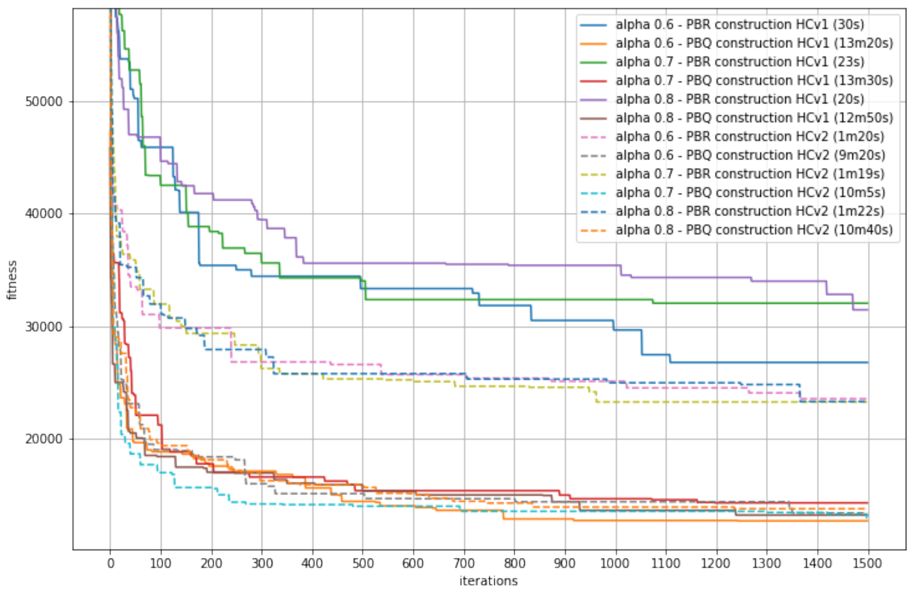


FIGURE 5.4: Cost of best solution found by of GRASP with PBR and PBQ construction functions and both versions of HC algorithms

The final experiment of GRASP solver with PBR and PBQ construction heuristics set a stopping condition for GRASP as the execution time instead of a fixed number of iterations. For our purpose described in section 1.2 I set a time limit to 3 minutes. Figure 5.5 shows the measured results of different combinations of alpha, construction heuristic, and HC procedures. The best performing combination is the PBR + HCv2, and the worst-performing combination is the PBR + HCv1. Even though the combinations with PBQ construction heuristic were not able to explore wider search space (due to the time complexity of the PBQ procedure), the difference in the cost of the solutions found is minimal ($< 5\%$)

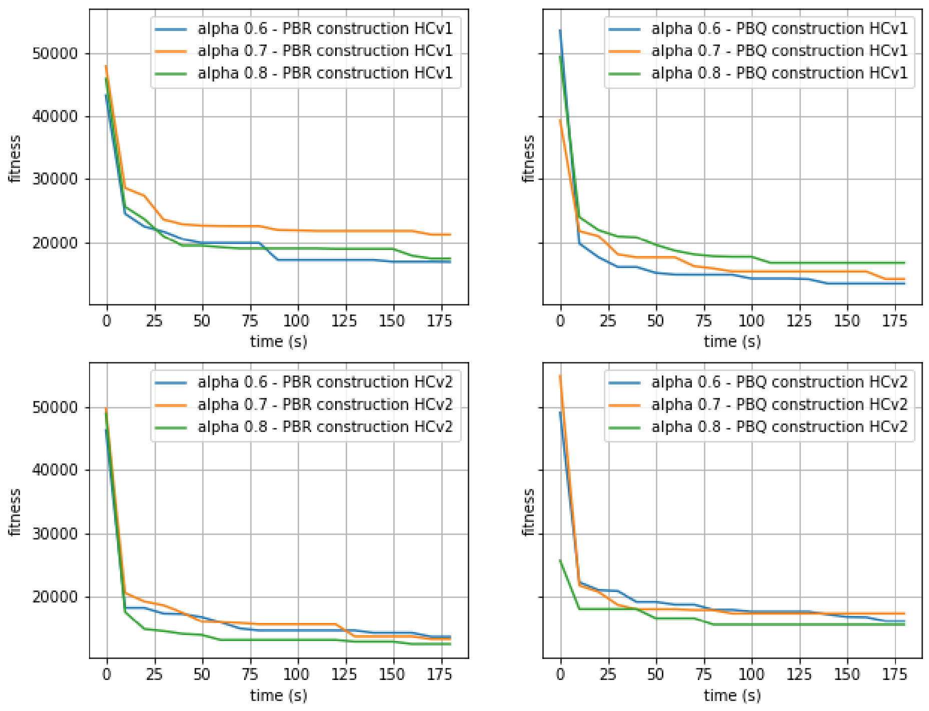


FIGURE 5.5: Cost evolution of PBR and PBQ constuction functions with 3 minute time limit on an instances with 100 customers and 10 vehicles

5.5.1 Comparing with exiting solvers

Based on all conducted measurements, I have decided to use a combination of PBQ + HCv1 and PBR + HCv2 as they have proven to be the best-performing ones. Solutions found by both versions of the GRASP solver are compared in the table 5.4 with other solves previously described in section 5.2.

Outcome of the measurements show that both versions of the GRASP solver are performing pleasingly. Average delivery time is lowest within compared solvers, and the average delay is somewhere between the results of HALNS and Jackpot solvers.

Solver	delay per order(min) (min/avg/max/median)	duration per order (min) (min/avg/max/median)	total distance traveled	total time spent	time (ms)
HALNS + Or-Tools	3.8/118.9/0	28.3/98.9/26	1844	14160	471
HALNS only	3.8/119/0	27.9/100/24.8	1844	13974	148
Jackpot + Or-Tools	3.7/128/0	28.3/179.7/23.9	2061	14181	497
Jackpot only	3.9/128/0	28.2/179.7/23.4	2062	14124	134
Insertion + Or-Tools	1.1/59/0	31.8/113/30.8	1707	15915	342
Insertion only	1.2/59/0	31.8/113/30.7	1714	15912	12
GRASP (PBQ) + Or-Tools	0.13/16.5/0	21.9/65/19.8	2093	10960	1192
GRASP (PBQ) only	0.8/25.7/0	21.8/69.3/19.7	2093	10904	870
GRASP (PBR) + Or-Tools	0.19/11.5/0	20/76.7/15.9	3607	10003	529
GRASP (PBR) only	1.4/30.8/0	19.6/89.3/15.7	3603	9799	186

TABLE 5.4: Result of different solvers on instances with 500 customers and 70 vehicles

Chapter 6

Conclusion

6.1 Summary

During the first part, I conducted experiments with different construction heuristics and compared them to my colleague's insertion heuristic [36]. Out of all the construction heuristics included in this thesis, the SEQ heuristic is the only one that doesn't apply to the problem I am confronted with. Nonetheless, it might be relevant to different VRP problems (e.g. VRP with an unlimited fleet). Among all implementations of construction heuristics, the PBR heuristic is the most efficient in terms of search space explored within the given time limit. The quality of solution provided by PBR are slightly worst then the one provided by PBQ constuciton heuristic. A disadvantage of the PBQ heuristic is its time complexity, but its solutions are consistently the best (lowest cost). The last construction heuristic, PIR, whose asymptotic complexity is the same as the complexity of PBQ, yielded the least desirable results of the other two heuristics (PBR and PBQ) for GRASP. Thus, this heuristic will not be used as a construction heuristic. In the table 5.2, the final results demonstrate there is not much difference in quality between PBR, PBQ, and insertion methods.

In the second part of the experiment, I investigated different combinations of construction heuristics with hill-climbing procedures, and alpha values to see what effect they had on the GRASP solver's performance. Combinations where the highest performance was achieved, include PBR+HCv2 and PBQ+HCv1. They both provided comparable results within a 3-minute time limit.

Different solvers have been compared with the GRASP implemented in this thesis, and the results are presented in table 5.4. The quality of the solutions provided by the GRASP with previously mentioned combinations of construction heuristic achieved similar results as Jackpot and HALNS solvers implemented in the thesis [36].

6.2 Future work

My future research will focus on developing a "better" (less greedy/randomized) local search technique for the VRP problem. As the local search used in this research has shown little to no improvement in the quality of solutions. Second, instead of adding a request at the end of the route, I will attempt to enhance the construction phase by inserting a request at the most promising position within the existing route. This adjustment, I believe, will significantly reduce the execution time for route construction. The HC procedure can then be called only during the Local search phase or if the request insertion yields an unfeasible route. Finally, the GRASP solver created in this thesis, along with other solvers, will be integrated into our GoDeliver's planning algorithm.

Bibliography

[1] Dantzig George B. and Fulkerson Delbert Ray. "MINIMIZING THE NUMBER OF CARRIERS TO MEET A FIXED SCHEDULE". In: *Naval Research Logistics Quarterly* 1 (1954), pp. 217–222 (cit. on p. 2).

[2] J. K. Lenstra and Rinnooy Kan A. H. G. "Complexity of vehicle routing and scheduling problems". In: *Networks* 11.2 (1981), pp. 221–227. DOI: `https://doi.org/10.1002/net.3230110211` (cit. on pp. 2, 11).

[3] L. Bodin et al. "Routing and Scheduling of Vehicles and Crews - The State of the Art". In: 10 (1983), pp. 63–212 (cit. on p. 5).

[4] R. Johnson and M. G. Pilcher. "The traveling salesman problem". In: vol. 19. 5. 1985, p. 463. DOI: `https://doi.org/10.1002/net.3230190511` (cit. on p. 2).

[5] M.W.P. Savelsbergh. "Local search in routing problems with time windows". In: *Annals of Operations Research* 4 (1985), pp. 285–305. DOI: `https://doi.org/10.1007/BF02022044` (cit. on p. 11).

[6] Desrosiers Jacques et al. "Methods for routing with time windows". In: *European Journal of Operational Research* 23.2 (1986), pp. 236–245. ISSN: 0377-2217. DOI: `https://doi.org/10.1016/0377-2217(86)90243-2` (cit. on p. 2).

[7] Solomon M. "Algorithms for the Vehicle Routing and Scheduling Problem with time Window Constraints". In: *Operations Research* 35 (1987), pp. 254–265. DOI: `http://dx.doi.org/10.1287/opre.35.2.254` (cit. on pp. 12, 17).

[8] Martin Desrochers and Francois Soumis. "A Generalized Permanent Labelling Algorithm For The Shortest Path Problem With Time Windows". In: *INFOR: Information Systems and Operational Research* 26.3 (1988), pp. 191–212. DOI: `10.1080/03155986.1988.11732063` (cit. on p. 2).

[9] LAPORTE GILBERT, NOBERT YVES, and TAILLEFER SERGE. "Solving a Family of Multi-Depot Vehicle Routing and Location-Routing Problems". In: *Transportation Science* 22.3 (1988), pp. 161–172. ISSN: 00411655, 15265447 (cit. on p. 5).

[10] Solomon Marius M. and Desrosiers Jacques. "Survey Paper—Time Window Constrained Routing and Scheduling Problems". In: *Transportation Science* 1 (1988), pp. 1–13 (cit. on p. 2).

[11] Martin Desrochers, Jacques Desrosiers, and Marius Solomon. "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows". In: *Operations Research* 40.2 (1992), pp. 342–354. DOI: `10.1287/opre.40.2.342` (cit. on p. 2).

[12] Laporte Gilbert. "The vehicle routing problem: An overview of exact and approximate algorithms". In: *European Journal of Operational Research* 59.3 (1992), pp. 345–358. ISSN: 0377-2217. DOI: `https://doi.org/10.1016/0377-2217(92)90192-C` (cit. on pp. 2, 5).

[13] Kokubugata H., Itoyama H., and Kawashima H. "Vehicle Routing Methods for City Logistics Operations". In: *IFAC Proceedings Volumes* 30.8 (1997), pp. 727–732. DOI: `https://doi.org/10.1016/S1474-6670(17)43907-3` (cit. on p. 5).

[14] Guy Desaulniers et al. "The Vehicle Routing Problem". In: Jan. 2002. Chap. VRP with Time Windows, pp. 157–193. ISBN: 978-0-89871-498-2. DOI: `10.1137/1.9780898718515.ch7` (cit. on p. 6).

[15] Guy Desaulniers et al. "The Vehicle Routing Problem". In: Jan. 2002. Chap. VRP with Pickup and Delivery, pp. 225–242. ISBN: 978-0-89871-498-2. DOI: `10.1137/1.9780898718515.ch9` (cit. on p. 6).

[16] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. Jan. 2002. DOI: `10.1137/1.9780898718515` (cit. on pp. 1, 2).

[17] Marco Diana and Maged M. Dessouky. "A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows". In: *Transportation Research Part B: Methodological* 38.6 (2004), pp. 539–557. ISSN: 0191-2615. DOI: `https://doi.org/10.1016/j.trb.2003.07.001` (cit. on pp. 19, 36).

[18] Gerardo Berbeglia et al. "Static pickup and delivery problems: A classification scheme and survey". In: *An Official Journal of the Spanish Society of Statistics and Operations Research* 15 (Feb. 2007), pp. 1–31. DOI: `10.1007/s11750-007-0009-0` (cit. on p. 2).

[19] Sophie Parragh, Karl Doerner, and Richard Hartl. "A survey on pickup and delivery problems: Part I: Transportation between customers and depot". In: *Journal für Betriebswirtschaft* 58 (Apr. 2008), pp. 21–51. DOI: `10.1007/s11301-008-0033-7` (cit. on p. 2).

[20] Eksioglu B., Vural A. V., and Reisman A. "The vehicle routing problem: A taxonomic review". In: *Computers & Industrial Engineering* 4 (2009), pp. 1472–1483 (cit. on p. 1).

[21] JY Potvin. "A Review of Bio-inspired Algorithms for Vehicle Routing". In: *Studies in Computational Intelligence* (2009), pp. 1–13. DOI: `https://doi.org/10.1007/978-3-540-85152-3_1` (cit. on p. 5).

[22] Hoff Arild et al. "Industrial aspects and literature survey: Fleet composition and routing". In: *Computers & Operations Research* 37.12 (2010), pp. 2041–2061. ISSN: 0305-0548. DOI: `https://doi.org/10.1016/j.cor.2010.03.015` (cit. on p. 1).

[23] El-Sherbeny Nasser A. "Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods". In: *Journal of King Saud University - Science* 22.3 (2010), pp. 123–131. ISSN: 1018-3647. DOI: `https://doi.org/10.1016/j.jksus.2010.03.002` (cit. on p. 6).

[24] Mauricio Resende and Celso Ribeiro. "Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications". In: vol. 146. Sept. 2010, pp. 283–319. DOI: `10.1007/978-1-4419-1665-5_10` (cit. on p. 25).

[25] Manar Hosny. "Vehicle Routing with Pickup and Delivery: Heuristic and Metaheuristic Solution Algorithms". PhD thesis. Jan. 2012. ISBN: 978-3-659-20258-2 (cit. on pp. 6, 15, 26, 34).

[26] Kenneth Sörensen and Fred Glover. "Metaheuristics". In: Jan. 2013, pp. 960–970. ISBN: 978-1-4419-1137-7. DOI: `10.1007/978-1-4419-1153-7_1167` (cit. on p. 21).

[27] Vigo Daniele and Toth Paolo. *Vehicle Routing: Problems, Methods, and Applications*. 2nd edition. SIAM-Society for Industrial and Applied Mathematics, 2014. ISBN: 1611973589 (cit. on pp. 2, 6).

[28] Ta-Yin Hu and Chin-Ping Chang. "A revised branch-and-price algorithm for dial-a-ride problems with the consideration of time-dependent travel cost". In: *Journal of Advanced Transportation* 49 (Nov. 2014). DOI: `10.1002/atr.1296` (cit. on p. 9).

[29] Grandinetti L. et al. "The Multi-objective Multi-vehicle Pickup and Delivery Problem with Time Windows". In: *Procedia - Social and Behavioral Sciences* 111 (2014), pp. 203–212. ISSN: 1877-0428. DOI: `https://doi.org/10.1016/j.sbspro.2014.01.053` (cit. on p. 11).

[30] Gilbert Laporte, Stefan Ropke, and Thibaut Vidal. "Vehicle Routing: Problems, Methods, and Applications". In: Nov. 2014. Chap. Chapter 4: Heuristics for the Vehicle Routing Problem, pp. 87–116. ISBN: 978-1-61197-358-7. DOI: `10.1137/1.9781611973594.ch4` (cit. on p. 11).

[31] Mauricio Resende and Celso Ribeiro. *Optimization by GRASP*. 2016. DOI: `https://doi.org/10.1007/978-1-4939-6530-4` (cit. on pp. 21, 22, 25, 26).

[32] Liao Tsai-Yun, Hu Ta-Yin, and Wu Yu-Wen. "A Time-dependent Vehicle Routing Algorithms for Medical Supplies Distribution Under Emergency". In: *OPERATIONS AND SUPPLY CHAIN MANAGEMENT* 10.3 (2017), pp. 161–169. ISSN: 1979-3561 (cit. on p. 5).

[33] Kenneth Sörensen, Florian Arnold, and Daniel Palhazi Cuervo. "A critical analysis of the "improved Clarke and Wright savings algorithm"". In: *International Transactions in Operational Research* 26.1 (2019), pp. 54–63. DOI: `https://doi.org/10.1111/itor.12443` (cit. on p. 11).

[34] *COVID-19 has changed online shopping forever, survey shows*. 2020. URL: `https://unctad.org/news/covid-19-has-changed-online-shopping-forever-survey-shows` (cit. on p. 2).

[35] *EUROPE FREIGHT AND LOGISTICS MARKET - GROWTH, TRENDS, COVID-19 IMPACT, AND FORECASTS* 2021 – 2026. 2020. URL: `https://www.mordorintelligence.com/industry-reports/european-freight-logistics-market` (cit. on p. 1).

[36] David Mokos. "Online Planner for Food Deliveries". PhD thesis. June 2020. URL: `https://dspace.cvut.cz/handle/10467/95468` (cit. on pp. 33, 34, 36, 45).

[37] Andrea Mor and M.Grazia Speranza. "Vehicle routing problems over time: a survey". In: *4OR* 18 (June 2020). DOI: `10.1007/s10288-020-00433-2` (cit. on p. 2).