Universidade de Aveiro Departamento de Matemática,
2023

Omoniyi Raymondjoy OBAFEMI

VEHICLE ROUTING PROBLEM WITH DELIVERY AND PICKUP

PROBLEMA DE PLANEAMENTO DE ROTAS DE VEÍCULOS COM ENTREGA E RECOLHA

Universidade de Aveiro Departamento de Matemática,
2023

Omoniyi Raymondjoy OBAFEMI

VEHICLE ROUTING PROBLEM WITH DELIVERY AND PICKUP

PROBLEMA DE PLANEAMENTO DE ROTAS DE VEÍCULOS COM ENTREGA E RECOLHA

Thesis presented to the University of Aveiro to fulfill the requirements to obtain the Master in Mathematics and Applications – Specialization in Statistics and Optimization, carried out under scientific guidance by Prof. Cristina Requejo, Professor of the Department of Mathematics of the University of Aveiro.

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do Mestrado em Matemática e Aplicações - Especialização em Estatística e Otimização, realizada sob orientação científica da Professora Cristina Requejo, docente do Departamento de Matemática da Universidade de Aveiro.

o júri / the jury

Presidente / President	**Professor Maria Raquel Rocha Pinto** Associate Professor, University of Aveiro
Vogais / Committee	**Professor Vladimir Protasov** Full Professor, University of L'Aquila
	Professor Cristina Requejo Associate Professor, University of Aveiro

agradecimentos / acknowledgements

I would like to express my gratitude to the following individuals and entities, who have been instrumental in the completion of this project.
Firstly, I offer my heartfelt appreciation to God Almighty for granting me the gift of life and sustenance throughout this journey.
Next, my deep appreciation goes to my wife, Juliana, for her unwavering strength and support in managing our family affairs while I pursued my studies abroad for two years. Your resilience and understanding is appreciated big thank you. I offer my love and gratitude to my children, Adesewa, Adewunmi, and Adebowale (Born when I was away).
Next to my sweet mother. May you live long to eat the fruit of your labour in Jesus name. And also to my Baby Sis Maryam, may the Almighty continue to guide you in you path, thanks for the encouragement dear.
I extend my sincere thanks to my Project Supervisor (Prof. Cristina Requejo) for your exceptional guidance and mentoring in the field of Optimization. Your insights and motherly advice have been invaluable in shaping the direction of this project and my life, I say Thank You.
I am grateful to the Director of the Mathematics Department for your generosity in providing me with a computer when my system encountered technical difficulties. Your swift assistance enabled me to continue my work without interruption. I wish to acknowledge the entire staff of the University of Aveiro for their contribution to creating a conducive academic environment. I extend my gratitude to Prof. Bruno Robino, the Coordinator of Intermaths, and the entire consortium of Intermaths for their support throughout the program. Special thanks to all the professors in the Intermaths program for their dedication to education.
I am indebted to Adebowale Ojo for your unwavering brotherly support and friendship, big thanks to you bro. To Folorunsho Sadiq, my school father, I am thankful for your fatherly love and guidance. A big thanks to Sir Prosper, Pastor Femi Lambe, Pastor Seun Aina, Taiwo Opatola, Aiyelagbe Samuel, Engineer BJ, Akinyemi Samuel, James Olatoke, Professor Halima Usman, Professor Magami, Mr. PJ Remember, Brother Gbenga Awolesi, Baba Jodi, Mr Olusegun, Mr. Adebayo (Aveiro), Mickysoft and Family, Lady Confo and husband Mr Kolade, Jide Ogundare, Egbon Obajamiu, Sister Keji, the MMU FBC Sokoto, Muyiwa Ogedengbe, TopsyJanded, and Mrs. Titilayo Obe and many others who have provided guidance and encouragement. My appreciation goes to the leadership of ADSU and the New Covenant Baptist Church Akute for your prayers, care, and support.
I extend my sincere appreciation to my classmate and well wishers the likes of Ashante Francsis, Oscar, Partrick, Hamed Senior, Agnes, Micheal(Mazi Obidient), Abel Maviano, Mathins Chinemerem(my personal person), bro Daniel David, sis Margret, sir Fanta, sir Breeze, Bro Uche(Roma, most senior bro), sir Futsum (Eritrean), sir Tijan Bah, Emma Oddoi, Betty, Sir Prosper (Guru Developer) and Awal(Zamani).
The completion of this program would not have been possible without your encouragement, assistance, and blessings. Thank you for being a part of my journey.

Keywords

Optimization; Routing; Logistics Planning; Transportation Optimization; Supply Chain Management; Delivery Scheduling; Pickup Scheduling; Mixed-Integer Programming (MIP).

Abstract

This project, on Vehicle Routing Problem with Delivery and Pickup (VR-PDP) aimed at determining the factors that contribute to the quality of a solution and checking the effect of varying the relative sizes of delivery demands to the size of pickup demands. This work was done with a dataset generated by a random uniform distribution of complete graphs with 15, 20, and 25 nodes, with varying relative sizes of the demands at each node to create four scenarios, Indifferent relative sizes of delivery to pickup, Large delivery demand relative to pickup demand, small delivery relative to pickup demand and delivery demand relatively equal to pick up demand at each customer. Three distinct problems were created using these scenarios. A model was created using flow formulation on the GurobiPy solver. The three problems were solved using the model and the result was tabulated. Observations on the table were thoroughly examined and relevant inferences were made on the factors that influence the quality of the VRPDP solution and the effect of varying the relative sizes of the demands

Palavras-chave

Otimização; Planeamento de Rotas; Planeamento Logístico; Otimização de Problemas de Transportes; Gestão da Cadeia de Abastecimento; Agendamento de Entregas; Agendamento de Recolhas; Programação Inteira Mista (MIP).

Resumo

Este projeto, sobre Problema de Planeamento de Rotas de Veículos com Entrega e Recolha (VRPDP), teve como objetivo estudar e determinar diversos fatores que possam contribuir para a qualidade de uma solução do problema, houve particular atenção em verificar o efeito que a variação nas quantidades de procura (quantidades a entregar) e de recolha têm nas soluções. Este trabalho foi realizado usando um conjunto de dados gerados aleatoriamente, através de uma distribuição uniforme, que permitiu construir grafos completos com 15, 20 e 25 vértices, gerar os seus custos de ligações e também gerar diferentes quantidades de procura e de recolha de forma a criar quatro cenários diferentes. Um cenário tem quantidades de procura e recolha sem qualquer relacionamento, outro cujos valores de procura são sempre superiores aos da recolha, outro cujos valores de recolha são sempre superiores aos da procura e, finalmente, o quarto cenário no qual os valores para entrega e recolha são iguais. São propostas formulações em Programação Inteira Mista para o Problema de Planeamento de Rotas de Veículos com Entrega e Recolha (VRPDP) usando diferentes conjuntos de restrições para formulações de fluxos. Um modelo foi construído, em Gurobipy, para ser usado no solver Gurobi. Os problemas foram resolvidos utilizando o modelo e os resultados obtidos para os diversos cenários foram resumidos em tabelas. As tabelas foram examinadas minuciosamente e inferências relevantes foram realizadas sobre os fatores que influenciam a qualidade da solução VRPDP e o efeito da variação das quantidades relativas das procuras e das recolhas.

Contents

1 Introduction **1**

1.1 Evolution of Vehicle Routing Problem 1

1.2 Objective 2

1.3 Outline 3

2 The Vehicle Routing Problem **5**

2.1 Variants of VRP 6

2.2 Importance and Challenges of Vehicle Routing Problems 8

2.3 Review on VRP as a MILP 9

2.4 Common Types of Formulations for the VRP 10

2.5 Formulation using MTZ constraints 13

2.6 Formulation using Commodity Flows 16

2.7 Some Work on Vehicle Routing Problems 18

3 The VRP with Pickup and Delivery **21**

3.1 Real Life Examples of the VRPDP 21

3.2 The problem formulation 22

3.3 Formulation of the VRPDP Using Commodity Flows 25

3.3.1 Example of VRPDP using Commodity Flow Formulation 27

3.4 Some Work on the VRP with Pickup and Delivery 29

4 Computational Experience **31**

4.1 Prototypical Example 32

4.2 Real Life Datasets 39

5 Conclusion and Future Work **45**

5.1 Summary Remarks 45

5.2 Future Works 45

Appendices **47**

A Appendix Table **48**
A.1 Problem 1, 15 Node, Indiffent Relative Delivery Demands to Pickup. 49
A.2 Problem 1, 15 Node, Large Relative Delivery Demands to Pickup. 50
A.3 Problem 1, 15 Node, small Relative Delivery Demands to Pickup. 51
A.4 Problem 1, 15 Node, Equal Relative Delivery Demands to Pickup. 52
A.5 Problem 2, 15 Node, Indiffent Relative Delivery Demands to Pickup. 53
A.6 Problem 2, 15 Node, Large Relative Delivery Demands to Pickup. 54
A.7 Problem 2, 15 Node, Small Relative Delivery Demands to Pickup. 55
A.8 Problem 2, 15 Node, Equal Relative Delivery Demands to Pickup. 56
A.9 Problem 3, 15 Node, Indiffent Relative Delivery Demands to Pickup. 57
A.10 Problem 3, 15 Node, Large Relative Delivery Demands to Pickup. 58
A.11 Problem 3, 15 Node, Small Relative Delivery Demands to Pickup. 59
A.12 Problem 3, 15 Node, Equal Relative Delivery Demands to Pickup. 60
A.13 Problem 1, 20 Node, Indiffent Relative Delivery Demands to Pickup. 61
A.14 Problem 1, 20 Node, Large Relative Delivery Demands to Pickup. 62
A.15 Problem 1, 20 Node, Small Relative Delivery Demands to Pickup. 63
A.16 Problem 1, 20 Node, Equal Relative Delivery Demands to Pickup. 64
A.17 Problem 2, 20 Node, Indiffent Relative Delivery Demands to Pickup. 65
A.18 Problem 2, 20 Node, Large Relative Delivery Demands to Pickup. 66
A.19 Problem 2, 20 Node, Small Relative Delivery Demands to Pickup. 67
A.20 Problem 2, 20 Node, Equal Relative Delivery Demands to Pickup. 68
A.21 Problem 3, 20 Node, Indiffent Relative Delivery Demands to Pickup. 69
A.22 Problem 3, 20 Node, Large Relative Delivery Demands to Pickup. 70
A.23 Problem 3, 20 Node, Small Relative Delivery Demands to Pickup. 71
A.24 Problem 3, 20 Node, Equal Relative Delivery Demands to Pickup. 72
A.25 Problem 1, 25 Node, Indiffent Relative Delivery Demands to Pickup. 73
A.26 Problem 1, 25 Node, Large Relative Delivery Demands to Pickup. 74
A.27 Problem 1, 25 Node, small Relative Delivery Demands to Pickup. 75
A.28 Problem 1, 25 Node, Equal Relative Delivery Demands to Pickup. 76
A.29 Problem 2, 25 Node, Indiffent Relative Delivery Demands to Pickup. 77
A.30 Problem 2, 25 Node, Large Relative Delivery Demands to Pickup. 78
A.31 Problem 2, 25 Node, Small Relative Delivery Demands to Pickup. 79
A.32 Problem 2, 25 Node, Equal Relative Delivery Demands to Pickup. 80
A.33 Problem 3, 25 Node, Indiffent Relative Delivery Demands to Pickup. 81
A.34 Problem 3, 25 Node, Large Relative Delivery Demands to Pickup. 82
A.35 Problem 3, 25 Node, Small Relative Delivery Demands to Pickup. 83
A.36 Problem 3, 25 Node, Equal Relative Delivery Demands to Pickup. 84

B Python codes **85**

B.1 Graph Generating Code 1. 86
B.2 Graph Generator 2 93
B.3 Graph Generating Code. 99

List of Figures

2.1 Some of the variants of the Vehicle Routing Problem. 7
2.2 Relative Load Sizing Constraints 15
2.3 Illustration of vehicle load constraints 15

3.1 Constraint on incoming vehicle to each node j. 23
3.2 Constraint on outgoing vehicle from each node j. 24
3.3 Constraint on Pickup Demand at node j 26
3.4 Constraint on Delivery Demand at node j 26
3.5 Solution to the Problem using Flow Formulation 29

4.1 Visualization of the Prototypical Dataset Graph 33
4.2 Solution of Optimal Routes Graph 37

List of Tables

2.1 Dimension of Problem with Increase in Numbers Nodes 13

3.1 Data of the example Using Generated Locations 27

3.2 Solution Using Commodity Flow Formulation 27

4.1 Position and demand details for prototypical Dataset 32

4.2 Distance Table for prototypical Dataset 33

4.3 Result summary on the prototypical example 38

4.4 Result summary on the 15 node problem 40

4.5 Result summary on the 20 node problem 41

4.6 Result summary on the 25 node problem 42

Chapter 1

Introduction

The Vehicle Routing Problem with Pickup and Delivery (VRPPD) is an extension of the classical Vehicle Routing Problem (VRP) where a fleet of vehicles is required to serve a set of customers with known demands. In the VRPPD, each customer may have both a pickup and a delivery request that must be served by the same vehicle with limited capacity, and the objective is to find a set of routes that minimizes the total cost usually taking into account the total distance traveled and the total number of vehicles used. The VRPPD is a practical and important problem that arises in a variety of contexts, such as public transportation, home health care, waste management, Courier services, food delivery, airline baggage handling, retail distribution, postal services, and cleaning services.

One of the main motivations for studying the VRPPD is that efficient routing of delivery vehicles, known as Vehicle Routing Problem with Pickup and Delivery (VRPPD), holds the potential to substantially decrease transportation costs, minimize fuel consumption, and lower carbon emissions. Simultaneously, it enhances service levels and customer satisfaction.

1.1 Evolution of Vehicle Routing Problem

Over time, solving the Vehicle Routing Problem (VRP) had being a tough puzzle. This problem has become even more complicated due to things like online shopping and delivering to customers' doors. We can see the changes in VRP by looking at the past, present, and future of the problem.

The origin of the Vehicle Routing Problem (VRP) can be traced back to the mid-20th century when distribution companies were faced with the task of optimizing delivery routes. The seminal work by George Dantzig and John Ramser in 1959 [8] marks a significant milestone in VRP's history. They introduced the "traveling salesman problem with a constraint" this layed the foundation for the subsequent evolution of the VRP. Dantzig's most significant contribution was the development of the "nearest neighbor" heuristic, an algorithm that starts with an arbitrary depot and repeatedly selects the closest unvisited customer until all customers are visited. This approach provided a simple yet effective solution for the VRP instances, and it has been widely used as a reference point for evaluating more sophisticated algorithms. Their pioneering algorithmic approach was initially applied to petrol deliveries, addressing the challenge of efficiently delivering goods from a central depot to customers who had placed orders. The central objective of the VRP remains the same till date, to minimize the overall cost of the delivery routes. This historical context showcases how the VRP emerged from practical distribution challenges into a well studied optimisation problem.

The importance of VRP grew because of global connections, business growth, and new technology in transportation. The rise of online shopping and the need for quick deliveries made VRP more are becoming more complex. It became crucial to meet customer needs accurately and on time, save money, and handle modern supply chains efficiently.

As VRP evolved, it inspired many ways to approach and solve these problems. Different methods like starting with groups or routes and using savings or insertion techniques have developed. Mathematics solvers, machine learning, and artificial intelligence are also used to find solutions to complex routing issues.

VRP is used in various fields. Businesses in retail, manufacturing, healthcare, and waste management. These fields use VRP to make their operations smoother, reduce wastage, and make their customers happier. It's also useful in city planning, responding to emergencies, and even guiding self driving cars. Today, VRP has adapted to new technologies. It now uses tools like maps, real-time data, and smart algorithms to plan routes better. Modern VRP mathematical solver can do many things like adjust routes in dynamic situations, plan for different capacities, consider sustainability, and even optimise loading in 3D.

Looking back at work done many years ago, it helps us understand how things were. Back then, there weren't computers or GPS like we have now. The research was different, but it still looked at how to make deliveries better. Even with these challenges, they have knowledge about the data of each path, math, and ways to make VRP work. They used real examples to see how things could be improved. Nowadays, VRP technology and solvers have changed a lot.

1.2 Objective

This project, Vehicle Routing Problem with Pickup and Delivery (VRPDP), is centered on using a mathematical solver (Gurobipy) to solve the problem. Exploring some instances and exploring the situation or scenario of relative size differences between the delivery demands and the pickup demands and also conducting a comprehensive comparison between the scenario's solution and the cost of each solution which demonstrates a deep understanding of both the problem domain and the optimal solution.

The utilization of the Gurobi solver is known for it efficiency and effectiveness in solving combinatorial problems, which VRPDP is part of, using matheuristics technique which employs majorly branch and cut to obtain the optimal solution or close to optimal solution of complex problems.

The aspect of varying the delivery demand and pickup relative to each other is to determine the effect of each demand's contribution to the complexity of solving the problem, hence the following will be considered

- Indifferent relative demands: here there are no restrictions on the delivery demand and the Pickup demands
- equal demands: here the delivery demands are required to be the same as the pickup demands;
- large demands: here the delivery demands are required to be large than the pickup demands at every location;
- small demands: here the delivery demands is restricted to be lesser than the pickup demands at every location.

The project goal is to compare the solutions across the various demand scenarios above and draw a meaningful inference about them.

1.3 Outline

This project comprises four additional chapters. Chapter two presents the Vehicle Routing Problem a quick review of what the Vehicle Routing Problem problem is. Chapter three, The Vehicle Routing Problem with Delivery and Pickup, is a quick review of this problem, its formulation, and approaches used in solving the problem. Chapter four, Computational Experience, contains the variety types of datasets used to implement a solution code written in Python using Gurobi solver, the tabulated result for different scenarios for easy study of the effect of some factors such as the vehicle capacity, relative quantity of pickup to delivery demands on the optimal cost. And Chapter five is the recommendation, conclusion, and summary.

Chapter 2

The Vehicle Routing Problem

In order to comprehend the Vehicle Routing Problem with Delivery and Pickup, it is essential to delve into the concept of the Vehicle Routing Problem, its variants, its nature, its formulation, and approaches used in solving them.

The Vehicle Routing Problem (VRP) entails the allocation of a fleet of vehicles to serve a group of customers, aiming to minimize the cost of the total distance traveled by the vehicles. The VRP holds great significance in the fields of transportation and logistics, and it also has high value and significant attention in recent research studies such as resource allocation problems, and logistics work. Then upon the foundation of the Vehicle Routing Problem (VRP) that we want to establish in this section, we will build the concept of Vehicle Routing Problem with Delivery and Pickup (VRPDP) which will be discussed in the next chapter, which extends the VRP by incorporating more specific constraints on it to make it more specific for Vehicle Routing Problem with Delivery and Pickup. This exploration will allow us to gain a deeper understanding of the challenges and complexities associated with the VRP and also its related variations.

The Vehicle Routing Problem (VRP) is widely recognized as an extension of the Traveling Salesman Problem (TSP) due to their inherent similarities but additional complexities such as a fleet of vehicles and possibilities to have multi-route is introduced in the VRP which brings about the differences. Several studies have explored the relationship between these two combinatorial optimization problems and established the VRP as a generalization of the TSP.

One of the works on this topic is the paper titled "A comparison of heuristics for the Vehicle Routing Problem" by Solomon (1987) [20], the author investigates various heuristics for solving the VRP and highlights the TSP as a special instance of the VRP when there is only one vehicle and no capacity constraints. Solomon's work emphasizes the broader applicability of VRP formulations by showing how they encompass the classic TSP.

In another study, "The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms" by Toth and Vigo (2002) [22]. The authors provided a comprehensive review of the VRP and highlighted its connection to the TSP. They discuss how the VRP encompasses the TSP as a special case by introducing multiple vehicles, capacity constraints, and other practical considerations.

The cited literatures above and some others, consistently affirm that the VRP is an expansion of the TSP.

As the number of customers increases it affects the difficulty of solving the VRP and consequently its computational time. It is well known that VRP is an NP-hard problem. NP-hardness, a fundamental concept in computational complexity theory, allows us to compare and classify the difficulty levels of different computational problems. It helps us identify problems that are at least as challenging as the toughest problems in the class of NP, which represents nondeterministic polynomial time. Since it is an established fact that VRP is an NP-hard problem, it signifies that finding exact optimal solutions for all VRP instances is computationally demanding and often impractical. The computational requirements for solving NP-hard problems, including VRP, grow exponentially as the problems sizes increase. Consequently, using exact algorithms to solve large VRP instances becomes impracticable due to the extensive computation time they require. Instead of focusing solely on optimality, approximation techniques become valuable as we are concerned with generating high-quality solutions that are frequently satisfactory in practical scenarios. By employing approximation techniques such as heuristics, matheuristics or metaheuristics, we can overcome the computational challenges associated with NP-hard problems and effectively produce solutions that are nearly optimal or meet the desired criteria. The reduction shows that VRP can be transformed into the well-known NP-hard problem, the Traveling Salesman Problem (TSP). This reduction allows to demonstrate that efficiently solving VRP would imply efficiently solving TSP. Since TSP is also NP-hard, this reduction confirms that VRP shares the same NP-hard classification with it.

2.1 Variants of VRP

The variants of the Vehicle Routing Problem encompass customized models that aim to precisely address specific real-life scenarios by incorporating special constraints into the classical vehicle routing problem. There are so many variants of the vehicle routing problem but some of the main ones will be highlighted below. The Figure 2.1 illustrates some of the main variants of the Vehicle Routing Problem (VRP), highlighting the different variations that exist within this problem domain. Each of these variants is described below.

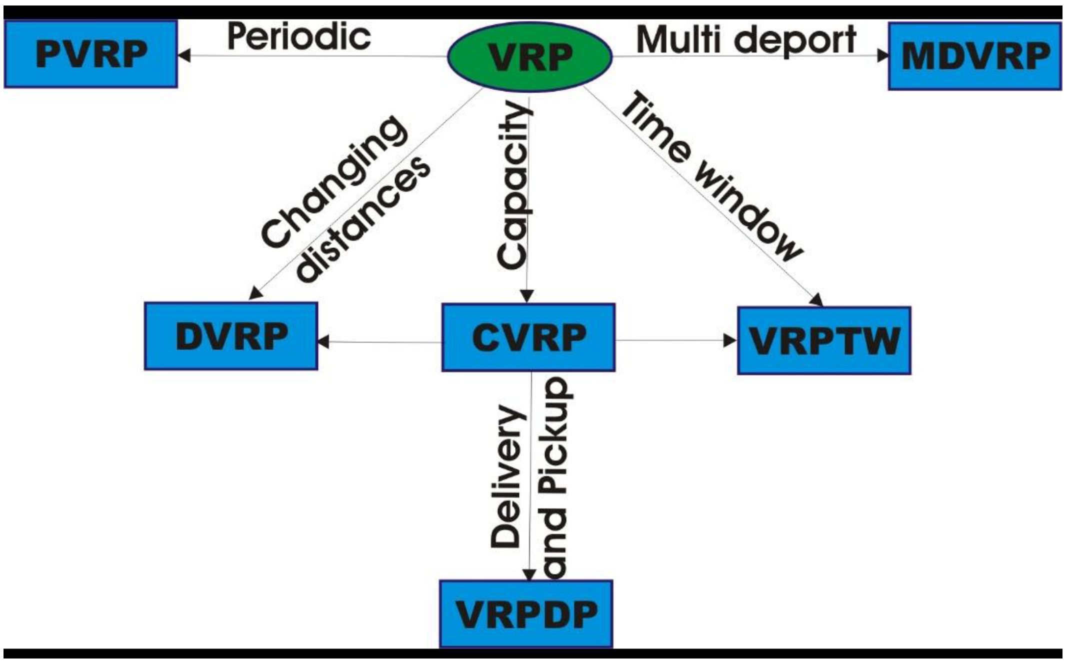

Figure 2.1: Some of the variants of the Vehicle Routing Problem.

Capacity Vehicle Routing Problem (CVRP). In this variant, each vehicle has a maximum capacity, and the total demand of the customers assigned to a vehicle should not exceed its capacity. This variant incorporates the capacity of the vehicle into the VRP.

Vehicle Routing Problem with Delivery and Pickup (VRPDP). This variant, involves pickup and delivery tasks, where goods or items need to be picked up from certain locations and delivered to others. Therefore, it incorporates both delivery and pickup operations in addition to considering the capacity of the vehicle.

Vehicle Routing Problem with Time Windows (VRPTW). In this variant, time windows are imposed on customer visits, specifying the time interval during which a customer can be served, this is accomplished by incorporating a time windows constraint for every customer.

Vehicle Routing Problem with Multiple Depots (VRPMD). In this variant, there are multiple depots or starting points for the vehicles. VRPMD extends the VRP by incorporating the presence of multiple depots, which introduces the need to allocate vehicles to depots and plan routes that satisfy the capacity constraints.

Periodic Vehicle Routing Problem (PVRP). In this variant, the objective is to determine optimal routes for a fleet of vehicles over a given planning horizon, where the routes repeat periodically.

Dynamic Vehicle Routing Problem (DVRP). In this variant it introduces the element of dynamic changes in customer demands and/or service locations over time. The dynamic Vehicle Routing Problem (DVRP) extends the VRP by considering real-time changes in customer demands, vehicle availability, and other dynamic factors. It aims to adapt the vehicle routes dynamically to optimize resource allocation and provide efficient service in response to evolving conditions throughout the course of the day.

2.2 Importance and Challenges of Vehicle Routing Problems

As already said, the Vehicle Routing Problem (VRP) is a fundamental and highly impacting combinatorial optimization challenge in the field of logistics and supply chain management. As businesses strive for greater efficiency, reduced costs, climate consciousness, and improved customer satisfaction, the significance of VRP becomes ever more pronounced. Some of the importance of a VRP and its solutions includes:

Complexity Challenges. VRP poses significant difficulties as it falls under the realm of combinatorial optimization, requiring to explore a vast number of possible combinations. The problem's degree of complexity arises from factors such as an increase in the number of customers, multiple constraints, varying vehicle capacities to model the real life scenario, and diverse customer demands.

Economic Significance. The logistics sector plays a vital role in the economy of many countries. In Australia, logistics activities account for approximately 8.6% of the total economy according to Machship (2022) [11]. Similarly, according to a report from 2021, Portugal witnessed a notable rise in the export of goods, with an 18.3% increase, and a corresponding 22.0% increase in imports. Additionally, the internal logistics within the country also experienced significant growth [9]. In Nigeria, the logistics sector contributes approximately 9.13% to the country's economy as reported by International Trade Administration (2023) [23]. Moreover, logistics costs can account for over 10% of the selling price of a commodity, underscoring the importance of efficient vehicle routing [16].

Environmental Sustainability. VRP can contribute to environmental sustainability by reducing fuel consumption and minimizing carbon emissions. Optimized routing strategies help decrease the number of vehicles on the road, leading to lower fuel consumption and environmental impact [18].

Improved Customer Service. By efficiently allocating resources and optimizing routes, VRP enables businesses to enhance their delivery schedules, reduce lead times, and improve overall customer service. Prompt and reliable deliveries are crucial for customer satisfaction and retention [7].

Urban Traffic Management. With the growing challenges of urban congestion, VRP plays a crucial role in managing traffic flows efficiently. By optimizing delivery routes and schedules, VRP can contribute to reducing traffic congestion and improving overall traffic management in urban areas [4].

2.3 Review on VRP as a MILP

Our focus is to considering the Vehicle Routing Problem (VRP) within the context of a Mixed Integer Linear Program (MILP) formulation. By examining the VRP through an MILP framework, we can uncover the potential for representing and analyzing this problem that incorporates both continuous and discrete decision variables.

Mixed Integer Linear Programming (MILP) is an optimization technique used in operations research and mathematical programming. It is an extension of Linear Programming (LP) where some of the decision variables are constrained to be integers rather than continuous values. It is usually by Branch and Bound algorithm, which explores the search space by branching on integer variables and bounding the solution space based on linear relaxation of the problem. MILP is a powerful tool for tackling real-world optimization problems that involve both continuous and discrete decision variables, and it has widespread use in various industries and research fields.

The formulation of the Vehicle Routing Problem (VRP) provides a structured representation that allows for the application of optimization techniques to achieve optimal or near-optimal solutions. A formulation that encompasses various components, such as decision variables, an objective function, and constraints, which effectively capture the fundamental requirements and limitations of the VRP.

To demonstrate the feasibility of representing the VRP as a Mixed Integer Linear Program (MILP), we need to carefully examine and analyze the objective function(s), decision variables, constraints, and restrictions involved in the MILP formulation. Through this analysis, we can illustrate how these elements can be applied to effectively model the VRP within the MILP framework.

Incorporation of Decision Variables. In the VRP formulation, the introduction of decision variables. the decision variables are the unknown quantities that we seek to determine in order to achieve the desired objective. These variables represent the choices or decisions we make to optimize a particular system or problem.

Incorporation of Objective Function. The formulation includes an objective function that quantifies the optimization goal, typically represent various goals, depending on the problem context. For example, in a production planning problem, the objective function might represent maximizing profit, minimizing costs, or maximizing production output, while in VRP it is the it minimizes the total cost, distance, or time required to serve all customers while satisfying certain constraints. The cost can represent various factors such as travel distance, travel time, vehicle operating costs, or a combination of these.

Incorporation of Constraints. Constraints play a crucial role in the formulation as they capture the requirements and limitations of the problem. These constraints encompass vehicle capacity constraints, flow into node constraint, flow out of node constraint, constraints on source node constraints; admissible flow in and out of the source node. By incorporating these constraints, the formulation ensures that the resulting vehicle routes satisfy the necessary conditions imposed by the problem.

By assuring the incorporation of decision variables, an objective function, and constraints, the VRP formulation establishes a structured representation that allows for the application of optimization techniques and effectively captures the essential aspects of the problem.

By allowing a mix of integer and continuous decision variables (real numbers) this make the above concept to be a mixed integer linear problem.

2.4 Common Types of Formulations for the VRP

Let a network be represented by a complete bidirected graph $G = (V, E)$, where $V = \{1, \ldots, n\}$ is the set of nodes of the graph G and corresponds to the locations of the customers. Set $E = \{(i, j)| i, j \in V, i \neq j\}$ is the set of arcs connection node i and node j and each arc is associated with a nonnegative cost c_{ij}. We additionally define s as the source node and N as the non-source nodes given by $N = V \backslash \{s\}$ Let $q_i > 0$ represent the demand of customer $i \in N$, and Q the capacity of the vehicles. Let K be the set of available vehicles, and we assume that the size of K is not predetermined before the formulation.

In order to establish an initial formulation for addressing this problem, we introduce binary variables x_{ijk}, where $(i, j) \in E$ and $k \in K$. These variables take on the value 1 when vehicle k visits customer j immediately after serving customer i, while assuming the roles of handling both delivery and pickup demands. Conversely, their value is 0 in all other cases. This representation aligns naturally with the solution's underlying structure.

$$\text{minimize} \quad \sum_{k \in K} \sum_{(i,j) \in E} c_{ij} x_{ijk} \tag{2.1}$$

$$\text{subject to:} \quad \sum_{k \in K} \sum_{i \in V \setminus \{j\}} x_{ijk} = 1, \quad \forall j \in N, \tag{2.2}$$

$$\sum_{j \in N} x_{sjk} \leq 1, \quad \forall k \in K, \tag{2.3}$$

$$\sum_{i \in V \setminus \{j\}} x_{ijk} = \sum_{i \in V \setminus \{j\}} x_{jik}, \quad \forall j \in N, \forall k \in K, \tag{2.4}$$

$$\sum_{i \in N} q_i \sum_{j \in V \setminus \{i\}} x_{ijk} \leq Q, \quad \forall k \in K, \tag{2.5}$$

$$\sum_{(i,j) \in S} x_{ijk} \leq |S| - 1, \quad \forall S \subseteq N, s \in S, \forall k \in K, \tag{2.6}$$

$$x_{ijk} \in \{0,1\}, \quad \forall (i,j) \in E, \forall k \in K. \tag{2.7}$$

- Expression (2.1) is the objective function that aims to minimize the total cost by summing the cost associated with arcs $(i,j) \in E$ over all vehicles $k \in K$.

- Equations (2.2) ensure that each customer j is visited exactly once by any vehicle k. For each customer j, the sum of decision variables x_{ijk} (excluding x_{jjk}) for all vehicles k and nodes $i \in V,\ i \neq j$ is equal to 1.

- Inequalities (2.3) state that each vehicle k is allowed to start from the depot (node s) at most once. The sum of decision variables x_{sjk} leaving the depot for all customers $j \in N$ is limited to be less than or equal to 1.

- Equations (2.4) are the flow conservation constraints and ensure the flow is maintained throughout the network. For each vehicle, $k \in K$, and customer $j \in N$, the total flow to node j should be the same as the total flow from node j.

- Inequalities (2.5) ensure that each vehicle's capacity is respected. The delivery demand q_i for each customer i in the route of each vehicle is subject to the vehicle's capacity Q.

- Equation (2.6). Subtour elimination constraints prevent the formation of subtours. The sum of decision variables x_{ijk} for arcs (i,j) in subset S, with $s \in S$, is restricted to be less than the size of S minus 1.

- Equation (2.7). The decision variables x_{ijk} are binary, they can only take values of 0 or 1, indicating whether vehicle k uses arc (i,j) in the network E or not.

The formulation's dimension complexity depends on two factors which are number of customers and vehicles. We consider the number of customers $n = |N|$ and the number of vehicles $m = |K|$. To gain an understanding of the scale of this formulation, we can determine, based on m and n.

Number of decision variables x_{ijk}:

- each x_{ijk} represents whether vehicle k travels from node i to node j;
- there are K possible values for k as there are K available vehicles;
- there are n nodes, so there are n possible values for i and for j;
- thus there are $n \times n \times k$ variables x_{ijk}.

Total number of decision variables $= K \cdot N \cdot N$

Total number of constraints $= N(K+1) + K + (2^N - 1) \cdot K$

and here is the breakdown of the constraints in the formulation:

- Constraint (2.2) introduces a constraint for each customer j. Since there are N customers, there are N such constraints.
- Constraint (2.3) introduces a constraint for each vehicle k. Since there are K vehicles, there are K constraints.
- Constraint (2.4) introduces a constraint for each node j and each vehicle k. Since there are N nodes and K vehicles, there are $N \cdot K$ constraints.
- Constraint (2.5) introduces a constraint for each vehicle k. Since there are K vehicles, there are K constraints.
- Constraint (2.6) introduces constraints for all possible subsets of customers. There are $2^N - 1$ non-empty subsets for N customers, so there are $(2^N - 1) \cdot K$ constraints.
- Constraint (2.7) has no dependency on K or N.

Total number of constraints = Number of constraints in Constraints (2.2) to (2.6) + Number of constraints in Constraint (2.7)

Total number of constraints $N + K + N \cdot K + K + (2^N - 1) \cdot K + 0$

For a clearer understanding of the variation in the number of variables and constraints, let's consider an example. We construct a table for this example, recording the number of variables and constraints as a function of the number of customers, assuming the availability of three vehicles.

The Table 2.1 presents the variation in the number of variables and constraints of the VRP problem for different numbers of customers and using only three vehicles.

Table 2.1: Dimension of Problem with Increase in Numbers Nodes

number of customers	4	7	10	11	12	13	25
Decision variables	48	147	300	363	432	507	1875
Number of the constraints.	67	415	3115	6191	12339	24631	100663399

As we can observe, considering 25 customers and 3 vehicles, the number of constraints is already enormous. The dimension of this formulation is primarily due to the presence of sub-route elimination constraints, identified by 2.6. In this formulation, such constraints have a cardinality that grows exponentially with the number n of customers. This implies that solving the linear relaxation of the problem becomes practically impossible when n is very large. One possible way to overcome this disadvantage is to consider a limited subset of these constraints and add the rest only if necessary, through appropriate constraint separation procedures. The considered constraints can be relaxed in Lagrangian form or explicitly included in the linear relaxation of the problem.

Alternatively, in this process, we can replace the constraints (2.6) with a family of constraints with polynomial cardinality. In this phase, we will introduce two families of constraints. One of the families was proposed by Miller, Tucker, and Zemlin (MTZ) of the Traveling Salesman Problem (TSP) called the MTZ formulation, and another family of constraints involves the use of flows called the Commodity flow Formulation. For the definition of each of these families, we use additional variables that aid in constructing the constraints. Since they utilize additional sets of variables, the formulations to be presented are extended formulations.

Next, we will utilize each of the two families of constraints in two groups of formulations. One group of formulations uses the main decision variables $x_{ijk} \in \{0,1\}$, which take the value 1 if the arc $(i,j) \in E$ is used by vehicle $k \in K$ to obtain the solution and take the value 0 otherwise. Another group involves the main decision variables being binary variables x_{ij}, which take the value 1 if the arc $(i,j) \in E$ is part of the solution and take the value 0 otherwise. We can verify that the use of binary variables x_{ijk} allows us to immediately identify the route assigned to each vehicle k, while with the variables x_{ij}, the routes are not immediately associated with the vehicles.

2.5 Formulation using MTZ constraints

In this section, we will consider the family of constraints proposed by Miller, Tucker, and Zemlin (MTZ) in [12] to eliminate sub-tours in earlier formulations. We continue to use the binary variables x_{ijk} and introduce additional continuous variables u_{ik}, where $i \in N$ and $k \in K$, representing the load of vehicle k after serving customer i. The MTZ constraints are introduced as a replacement for constraint (2.6) and (2.6), and they can be described using the following expressions

$$u_{jk} \geq u_{ik} + q_j - Q(1 - x_{ijk}), \quad \forall i, j \in N, i \neq j, \forall k \in K, \tag{2.8}$$

$$q_i \leq u_{ik} \leq Q, \quad \forall i \in N, \forall k \in K. \tag{2.9}$$

Thus, we have the opportunity to restructure the formulation from Equation(2.1) to (2.7) incorporating the MTZ constraints from Equations (2.8) and (2.9) as substitutes for the sub-tour elimination constraints mentioned in Equations (2.5) and 2.6

$$\begin{aligned}
\text{minimize} \quad & \sum_{k \in K} \sum_{(i,j) \in E} c_{ij} x_{ijk} && (2.1)\\
\text{subject to:} \quad & \sum_{k \in K} \sum_{i \in V \setminus \{j\}} x_{ijk} = 1, \quad \forall j \in N && (2.2)\\
& \sum_{j \in N} x_{0jk} \leq 1, \quad \forall k \in K && (2.3)\\
& \sum_{i \in V \setminus \{j\}} x_{ijk} = \sum_{i \in V \setminus \{j\}} x_{jik}, \quad \forall j \in V, \forall k \in K && (2.4)\\
& u_{jk} \geq u_{ik} + q_j - Q(1 - x_{ijk}), \quad \forall i, j \in N, i \neq j, \forall k \in K && (2.8)\\
& q_i \leq u_{ik} \leq Q, \quad \forall i \in N, \forall k \in K && (2.9)\\
& x_{ijk} \in \{0, 1\}, \quad \forall (i,j) \in E, \forall k \in K && (2.7)
\end{aligned}$$

The binary variable x_{ijk} takes the value 1 if vehicle k travels from customer i to customer j, and 0 otherwise. To prevent the formation of subtours, the constraint 2.8 enforces that if vehicle k travels from customer i to customer j, the load after visiting j (u_{jk}) must be greater than or equal to the load after visiting i (u_{ik}) plus the demand at j (q_j), adjusted by a term that considers the capacity Q when the arc (i, j) is not used making the constraint redundant in this case.

Additionally, constraint 2.9 stipulates that the load of vehicle k after visiting customer i (u_{ik}) must fall within the range of the minimum pickup or delivery demand (q_i) and the maximum capacity (Q). This way, the constraint system ensures that the vehicle's load is sufficient to accommodate the goods being transported, while the MTZ constraints replace the sub-tour elimination constraints in the original formulation. These constraints can be illustrated with a diagrams Figure 2.2, as shown below In Figure 2.8 and Figure 2.8

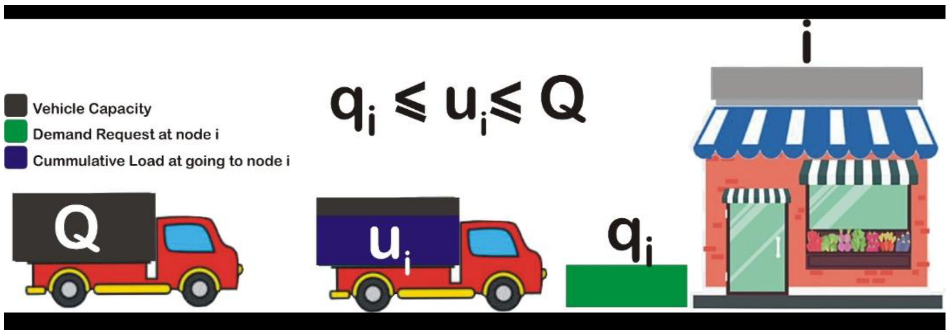
Vehicle Capacity
Demand Request at node i
Cummulative Load at going to node i
$q_i \leqslant u_i \leqslant Q$
i
Q
u_i
q_i

Figure 2.2: Relative Load Sizing Constraints

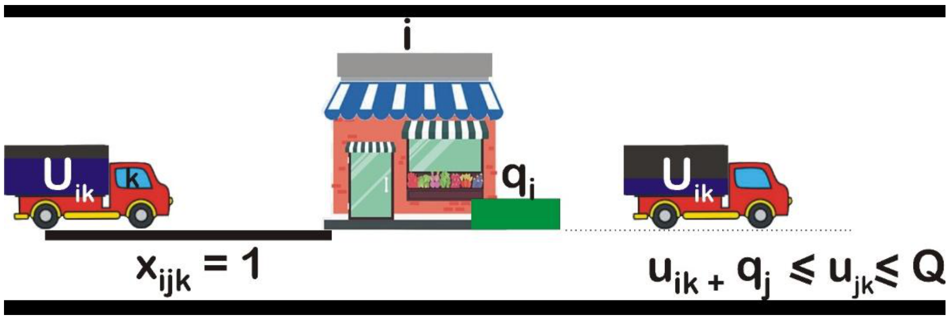

Figure 2.3: Illustration of vehicle load constraints

In Figure 2.2, we can observe that if customer i is visited by vehicle k, then the vehicle's load after visiting the customer falls between the customer's demand and the vehicle's capacity: $q_i \le u_{ik} \le Q$. If vehicle k travels from customer i to customer j which implies that $(x_{ijk} = 1)$, then the load after serving customer j is bounded between the vehicle's capacity and the sum of the load after serving customer i and the demand of customer j $(u_{ik} + q_j \le u_{jk} \le Q)$.

When vehicle k travels from customer i to customer j $(x_{ijk} = 1)$, according to Constraints (2.9) which is illustrated by Figure 2.3, we have $u_{jk} \ge u_{ik} + q_j$.

Just like we did earlier, let's figure out how many variables and constraints are in the MTZ formulation based on the values of n and m. This will help us compare the two formulations. In terms of n and m.

The number of variables x_{ijk} is $mn(n+1)$, and the number of variables u_{ik} is mn. Therefore, the number of variables for this formulation is defined by the following formula, Equation 2.10.

$$\text{Total number of decision variables} = m\ n\ (n+2) \tag{2.10}$$

The number of constraints for the formulation is defined by the following Equation 2.11

$$\text{Total number of constraints} = 2m + n + 3mn + 2mn^2 \tag{2.11}$$

It can be observed that the cardinality of the constraints now becomes polynomial in terms of m and n, unlike the sub-tour formulation which was an exponential that is the power of N.

In the following table, we record the variation in the number of variables and constraints based on the number n of customers, considering a fleet of three available vehicles.

Number of Customers	4	7	10	11	12	13	25
Number of Variables	72	189	360	429	504	585	2025
Number of Constraints	142	370	706	842	990	1150	4006

When we replace the sub-tour elimination rules 2.6 and the capacity limits (2.5) with the MTZ rules given by (2.8) and (2.9), we see that the number of rules becomes much smaller. For instance, if there are 3 vehicles and 25 customers, the number of constraints decrease from 100,663,399 to 4,006. This greatly helps in reducing the time it takes for calculations.

Lastly, we consider the formulation using commodity flow for the same problem.

2.6 Formulation using Commodity Flows

Another approach to handle the presence of sub-routes is by using flow restrictions to get rid of them. To do this, we can think of each vehicle k as carrying a certain amount of flow from the starting depot (node s) to every customer to fulfill their demands. To set up these restrictions, additionally to the binary variables x_{ijk} like before, now we introduce continuous variables f_{ijk} for all combinations of i and j in the set of nodes N where i is not equal to j, and for all vehicles k in the set of vehicles K. These new variables f_{ijk} represent the amount of flow transported from customer i to customer j using vehicle k. With this in mind, we can create another formulation for solving this problem using mixed-integer linear programming, as shown below.

$$\text{minimize} \quad \sum_{k \in K} \sum_{(i,j) \in E} c_{ij} x_{ijk} \tag{2.1}$$

$$\text{subject to:} \quad \sum_{k \in K} \sum_{i \in V \setminus \{j\}} x_{ijk} = 1, \quad \forall j \in N \tag{2.2}$$

$$\sum_{j \in N} x_{0jk} \leq 1, \quad \forall k \in K \tag{2.3}$$

$$\sum_{i \in V \setminus \{j\}} x_{ijk} = \sum_{i \in V \setminus \{j\}} x_{jik}, \quad \forall j \in V, \forall k \in K \tag{2.4}$$

$$\sum_{i \in V \setminus \{j\}} (f_{ijk} - f_{jik}) = q_j, \quad \forall j \in N, \forall k \in K \tag{2.12}$$

$$f_{ijk} \leq Q x_{ijk}, \quad \forall (i,j) \in E, \forall k \in K \tag{2.13}$$

$$f_{ijk} \geq 0, \quad \forall (i,j) \in E, \forall k \in K \tag{2.14}$$

$$x_{ijk} \in \{0,1\}, \quad \forall (i,j) \in E, \forall k \in K \tag{2.7}$$

The only differences from the previous formulation (Sub-tour Elimination Formulation) are introduction of new decision variables and replacement of constraints (2.5) and (2.6) by new constraints (2.12), (2.13) and (2.14). The functions of the new constraints are explained as follows.

Equation (2.12) is the Demand Satisfaction Constraint. This constraint ensures that the difference between the inflow and outflow at a customer vertex j (for vehicle k) equals the customer's demand q_j.

Inequality 2.13 is the Flow Capacity Constraint. This constraint limits the flow f_{ijk} on an arc (i,j) for vehicle k to be at most Q when vehicle k travels through arc (i,j) (as it is multiplied by the decision variable x_{ijk}), ensuring that the flow is within the capacity limits.

Non-Negativity of Flow Constraint (Inequality 2.14), this constraint ensures that the flow variables f_{ijk} on each arc (i,j) for vehicle k is non-negative, it restricts flow values to non-negative real numbers.

Similar to our previous formulations, we want to compute the count of variables and constraints in this commodity flow formulation.

The quantity of variables of type f_{ijk} matches the number of variables of type x_{ijk}, which is $mn(n+1)$, resulting in a total of variables to be $2mn(n+1)$

total constraint count expressed algebraically as $n + m + m(n+1) + mn + mn(n+1) + mn(n+1) + mn(n+1) = 3mn^2 + (5m+1)n + m$.

Just like we did with previous formulations, we will create a table to demonstrate how changing the value of N impacts the count of decision variables and constraints in the commodity flow formulation we're using.

Number of Customers	4	7	10	11	12	13	25
Number of Variables	120	336	660	792	936	1092	3900
Number of Constraints	220	565	1072	1277	1500	1741	6037

When comparing the three formulations, we observed that the sub-tour elimination approach is extremely costly to use. The commodity flow formulation, however, provides a more reasonable solution for solving the problem, as the constraint it employs is only about $\frac{3}{50000}$ of the sub-tour elimination constraint for $N = 25$. The MTZ (Miller-Tucker-Zemlin) formulation is even cheaper to use compared to the commodity flow formulation, with values of 4006 and 6037 respectively at $N = 25$.

We note that, when comparing the corresponding linear relaxations, the strength of the MTZ constraints compared to the flow formulation is weaker. This means that the performance of the MTZ formulation is poor when using solving approaches based on the linear programming relaxations of the formulations.

2.7 Some Work on Vehicle Routing Problems

The Vehicle Routing Problem (VRP) has been approached using various techniques, algorithms, and solvers in recent years. For small problem instances, exact mathematical optimization methods can be employed to find the optimal solution, ensuring its uniqueness. However, as the problem size grows, finding the exact optimal solution becomes computationally challenging. In such cases, heuristic, matheuristic, or meta-heuristic algorithms are often utilized to obtain near-optimal solutions within a reasonable time frame. More work done on VRP are examined and reported as follows.

The work by Tan and Yen (2021) [21] delved into the area of transportation planning, specifically focusing on vehicle routing problems (VRP) and their various real-world applications. With the rise in computational capabilities, the study highlights the increasing complexity of VRPs that can now be tackled using a range of algorithms. To better understand the recent advancements in this domain, they conducted a thorough analysis of literature published from 2019 to August 2021. They used a systematic approach, they classified both models and solutions into distinct categories, encompassing customer-related, vehicle-related, and depot-related models, as well as exact, heuristic, and meta-heuristic algorithms.

Jayarathna et al. (2019) [10] present an exploration of the contemporary concepts within logistics, focusing on innovative solutions for transportation and distribution systems. The paper's central goal is to motivate a in-dept thinking concerning Vehicle Routing Problems (VRP) and their evolution adaptations based on the fact that VRP is becoming more complex as the technologies evolve and more need for VRP arises. The authors spotlight essential VRP variants such as the Capacitated Vehicle Routing Problem (CVRP), Vehicle Routing Problem with Time Windows (VRPTW), and Vehicle Routing Problem with MultiDepot (MDVRP), recognizing their role in the broader VRP context. The study delves into recent developments, encapsulating discussions on VRP categorization, synthesizing common constraints, and constructing model algorithms. By undertaking a systematic approach, the paper examines the future implications of VRP modeling, forecasting that the realm of Intelligent Vehicle Routing Problems and Intelligent Heuristic Algorithms will emerge as pivotal domains for future research endeavors.

The work by Liong et al. (2008) considered the Vehicle Routing Problem (VRP) as well-studied operational research challenge involving the efficient distribution of supplies to customers from one or multiple depots. The work explored a range of approaches used to tackle the VRP. The paper highlights the application of heuristic methods such as genetic algorithms, evolution strategies, and neural networks, signaling the broadening scope of solution strategies. The beauty in this work is that the work provides a concise overview of the VRP's significance, its diverse problem variations, and the evolving packages of approaches researchers have harnessed to tackle its complexity for practical scenarios.

Chapter 3

The VRP with Pickup and Delivery

The Vehicle Routing Problem with Delivery and Pickup (VRPDP) is a combinatorial optimization problem that extends the classical Vehicle Routing Problem (VRP) by introducing both delivery and pickup tasks into the routing process. In the VRPDP, a fleet of vehicles is tasked with delivering goods to certain customer locations and picking up goods from other locations for transportation back to the depot or other specified destinations. The objective is to minimize the total distance traveled, total cost, or the number of vehicles used, while ensuring that all delivery and pickup tasks are completed within given constraints such as vehicle capacities and time windows.

3.1 Real Life Examples of the VRPDP

Various real-life examples demonstrate the practical significance of the VRPDP in numerous industries and logistics operations, where the efficient management of simultaneous delivery and pickup tasks is essential. Below, a few of these examples are explained to help understand the concept of the new tasks (Delivery and pickup) added.

- **Waste collection**: in urban waste collection, vehicles need to deliver empty bins to customer locations and pick up full bins for disposal. The VRPDP can optimize the routes to collect waste efficiently while minimizing the number of collection vehicles and distances traveled.

- **Reverse logistics**: in e-commerce or retail industries, vehicles may need to deliver new products to customers and simultaneously pick up returned items for refurbishment or recycling. The VRPDP can optimize the return routes, considering capacities and time windows, to handle reverse logistics efficiently.

- **Home healthcare services**: in mobile healthcare providers, such as nurses or caregivers, may need to deliver medical supplies to patients and pick up medical equipment or samples for testing. The VRPDP can optimize their routes to minimize travel time and ensure timely delivery and pickup.

- **Food delivery services**: in food delivery companies often face the challenge of simultaneously delivering prepared meals to customers and picking up empty containers from previous orders. The VRPDP can optimize the food delivery routes, considering delivery and pickup tasks to minimize overall transportation costs.

3.2 The problem formulation

Similar to the VRP discussed in the preceding chapter, the VRPDP variant also offers a variety of formulations. In this context, we will delve into the fundamental formulation, the commodity flow formulation, in this project, we would employ the directed flow formulation to optimize the vehicles' routes.

Let a network be represented by a complete bidirected graph $G = (V, E)$, where $V = \{1, \dots, n\}$ is the set of nodes of the graph G and corresponds to the locations of the customers. Set $E = \{(i, j) | i, j \in V, i \neq j\}$ is the set of arcs connection node i and node j and each arc is associated with a nonnegative cost c_{ij}. We additionally define s as the source node and N as the non-source nodes given by $N = V \backslash \{s\}$. We additionally have the following parameters:

- $q_i > 0$ represents the delivery demand of customer $i \in N$, and
- p_i stands for the pickup demand at node i.
- Q stands for the vehicle's capacity. It is assumed that the fleet of vehicles has the same capacity.

In order to establish an initial formulation for addressing this problem, we introduce binary variables x_{ij}, where $(i, j) \in E$. These variables take on the value 1 when the vehicle visits customer j immediately after serving customer i, while assuming the roles of handling both delivery and pickup demands. Conversely, its value is 0 in all other cases.

The following is the mathematical model for the VRPDP

$$\text{minimize} \quad \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{3.1}$$

$$\text{subject to:} \quad \sum_{i\in V\setminus\{j\}} x_{ij} = 1, \quad \forall j \in N \tag{3.2}$$

$$\sum_{i\in V\setminus\{j\}} x_{ji} = 1, \quad \forall j \in N \tag{3.3}$$

$$\sum_{j\in N} x_{sj} \le 1 \tag{3.4}$$

$$\sum_{i\in V\setminus\{j\}} x_{ij} = \sum_{i\in V\setminus\{j\}} x_{ji}, \quad \forall j \in N \tag{3.5}$$

$$\sum_{i\in N}\left((q_i - p_i)\sum_{j\in V\setminus\{i\}} x_{ij}\right) \le Q \tag{3.6}$$

$$\sum_{(i,j)\in S} x_{ij} \le |S| - 1, \quad \forall S \subseteq V \tag{3.7}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E \tag{3.8}$$

- Equation (3.1): the objective function aims to minimize the total cost by summing the cost associated with arcs (i,j) over all the arcs $(i,j) \in E$.

- Equation (3.2): this constraint ensures that each customer j is visited exactly once by a vehicle on a particular route. The Figure 3.1 illustrates this constraint.

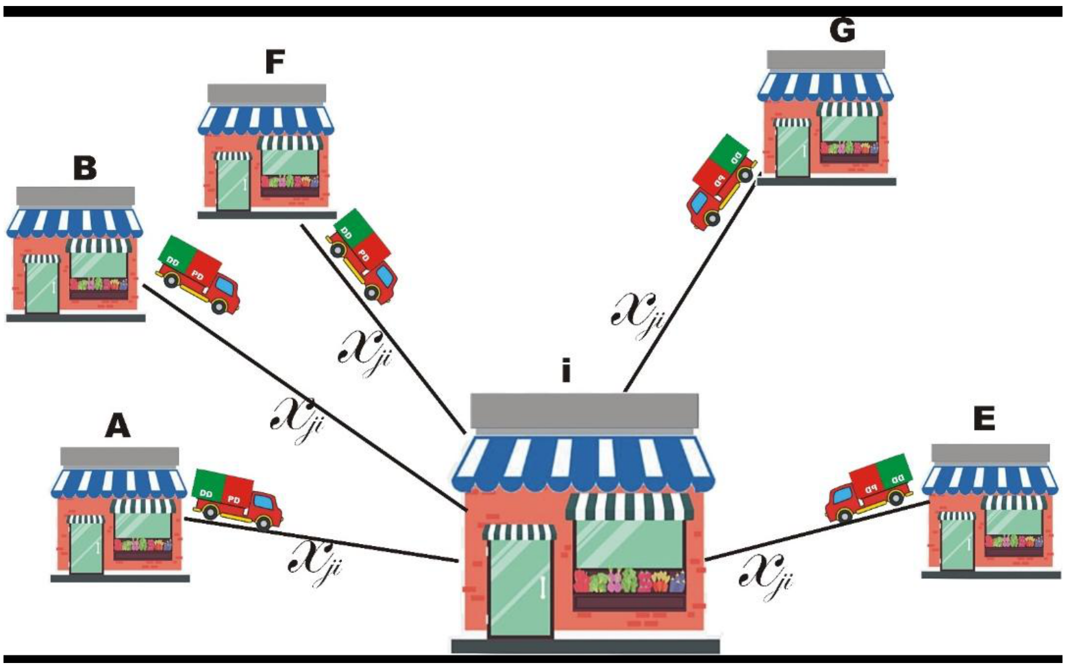

Figure 3.1: Constraint on incoming vehicle to each node j.

- Equation (3.3): This constraint ensures that each customer j is exited exactly once by a vehicle on a particular route. The Figure 3.2 illustrates this constraint.

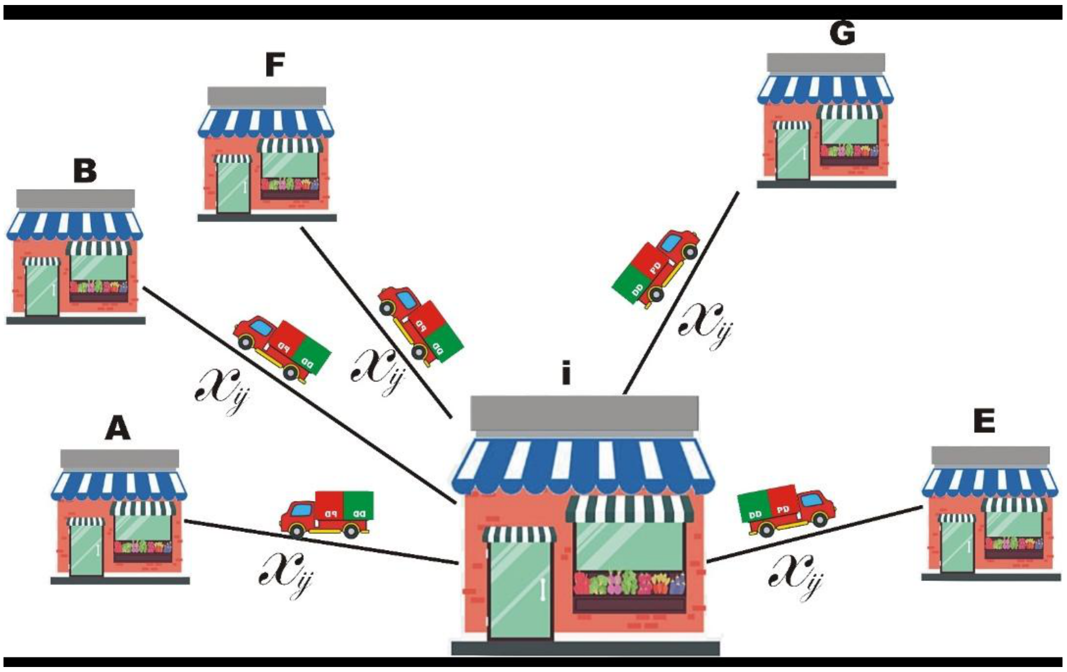

Figure 3.2: Constraint on outgoing vehicle from each node j.

- Equation (3.4): ensures that at most 1 vehicle is allowed to start from the depot (node s).

- Equation (3.5): ensures the flow conservation, is maintained throughout the network. For every node $j \in N$, the total flow to node j should be the same as the total flow from node j.

- Equation (3.6): ensures that the vehicle's capacity is complied to. The sum of the difference between the delivery demand q_i and pickup demand p_i for each customer i, multiplied by the corresponding flow variables x_{ij}, is less than or equal to the vehicle's capacity Q.

- Equation (3.7): ensures sub-tour elimination constraints to prevent the formation of sub-tours on the vehicle tour.

- Equation (3.8): ensures that the decision variables x_{ij} are binary, indicating whether a vehicle uses arc (i, j) in the network or not. They can only take values of 0 or 1.

3.3 Formulation of the VRPDP Using Commodity Flows

As discussed earlier in the last chapter, another approach to handle the presence of sub-routes is by using flow restrictions to get rid of them. To do this, we can think of a vehicle carrying a certain amount of flow from the starting depot (node s) to every customer to fulfill the demands of each customer. To set up these restrictions, we use binary variables x_{ij} like in the last section, and now we introduce continuous variables y_{ij}, z_{ij} for all combinations of i and j in the set of nodes N where i is not equal to j. These new variables y_{ij} represent the amount of delivery demand flow transported from node i to node j and z_{ij} represents the amount of pickup demand flow transported from node i to node j. With this in mind, we model this problem using mixed-integer linear programming, as shown below:

$$\text{minimize} \quad \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{3.1}$$

$$\text{subject to:} \quad \sum_{i\in V\setminus\{j\}} x_{ij} = 1, \quad \forall j \in N \tag{3.2}$$

$$\sum_{i\in V\setminus\{j\}} x_{ji} = 1, \quad \forall j \in V \tag{3.9}$$

$$\sum_{i\in V\setminus\{j\}} (y_{ij} - y_{ji}) = q_j, \quad \forall j \in N \tag{3.10}$$

$$\sum_{i\in N} y_{is} = 0, \tag{3.11}$$

$$\sum_{i\in V\setminus\{j\}} (z_{ij} - z_{ji}) = -p_j, \quad \forall j \in N \tag{3.12}$$

$$\sum_{i\in N} z_{si} = 0, \tag{3.13}$$

$$y_{ij} + z_{ij} \le Qx_{ij}, \quad \forall (i,j) \in E \tag{3.14}$$

$$y_{ij} \ge 0, \quad \forall (i,j) \in E \tag{3.15}$$

$$z_{ij} \ge 0, \quad \forall (i,j) \in E \tag{3.16}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E \tag{3.8}$$

Constraints (3.2) and (3.9) are equivalent to (3.5). The main differences from the previous formulation (Sub-tour Elimination Formulation) are the introduction of new decision variables and replacement of constraints (3.6) and (3.7) by new constraints (3.10), (3.11) (3.12), (3.13), (3.14), (3.15), and (3.16). The new constraints are explained as follows.

The pickup demand satisfaction constraint is considered in Equation (3.12), this constraint ensures that the difference between the inflow of pickup demanded and outflow of pickup demanded at a customer vertex j meets the customer's delivery demand p_j. This is illustrated in Figure 3.3

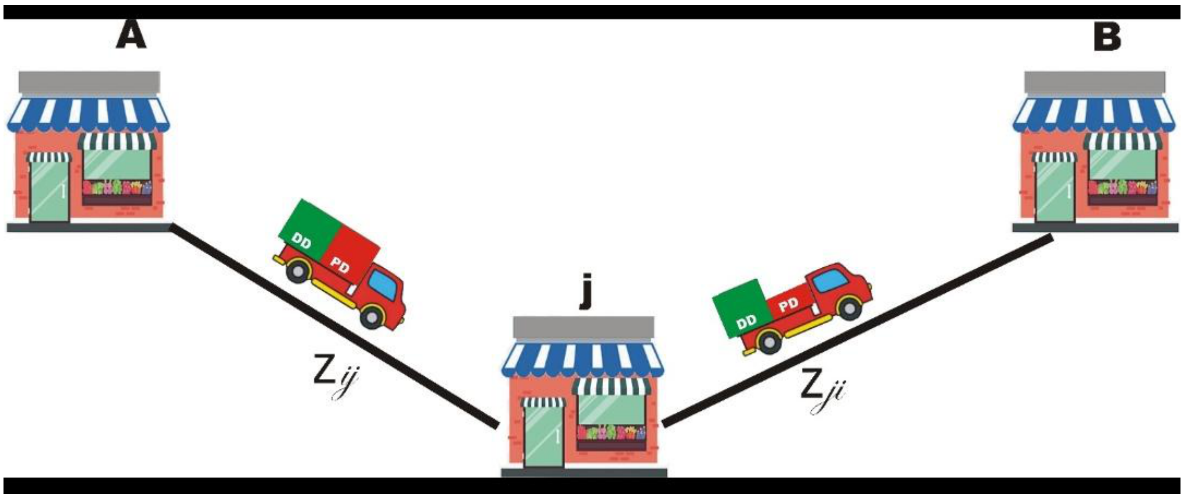

Figure 3.3: Constraint on Pickup Demand at node j

The delivery demand satisfaction constraint is considered in Equation (3.10), this constraint ensures that the difference between the inflow of delivery demanded and outflow of delivery demanded at a customer j meets the customer's delivery demand q_j. And this constraint is illustrated by Figure 3.4

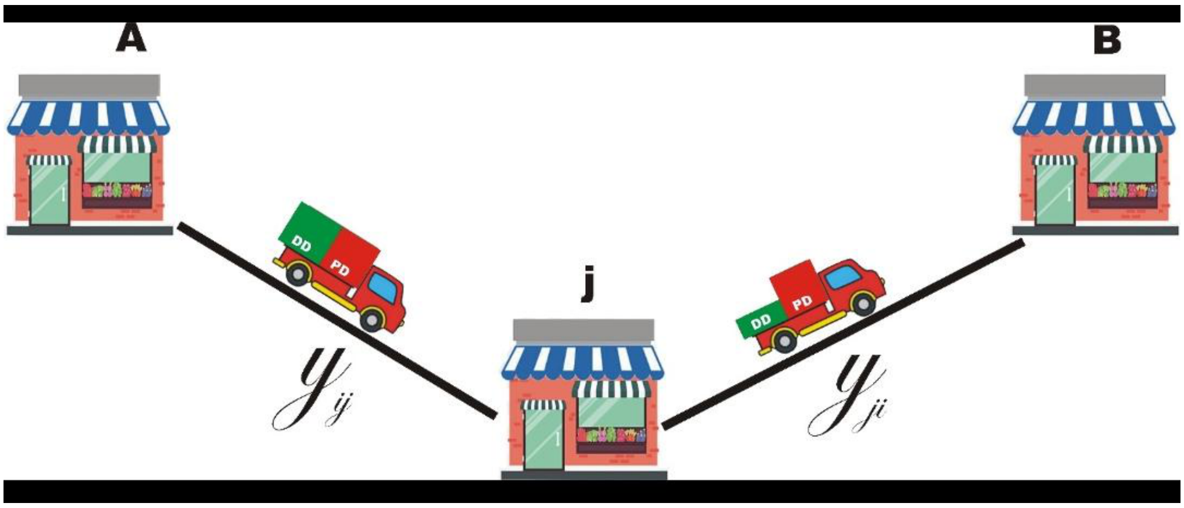

Figure 3.4: Constraint on Delivery Demand at node j

Constraints (3.11) and (3.13) ensure the inflow to the depot of the delivery demand is zero and the outflow of the pickup demand to the depot is also zero.

Resultant flow capacity constraint Equation (3.14), this constraint limits the sum of flow y_{ij}

and z_{ij} on an edge (i, j) must be at most Q, multiplied by the decision variable x_{ij}, ensuring that the flow is within the capacity limits.

Non-Negativity of demand flow constraint Equation (3.15) and Equation (3.16), these constraints ensure that the flow variable y_{ij} and z_{ij} on each edge (i, j) is non-negative, it restricts both demand flow values to nonnegative real numbers.

3.3.1 Example of VRPDP using Commodity Flow Formulation

We will solve a problem using this flow formulation and visualize the solution of the problem. The next table displays the problem data of the example.

Table 3.1: Data of the example Using Generated Locations

	pos_x	pos_y	demand_delivery	demand_pickup
Lisbon	1.46	0.95	0	0
Porto	70.55	34.97	5	4
Braga	80.00	12.13	5	1
Faro	0.46	65.88	13	4
Coimbra	9.47	0.23	10	11
Aveiro	0.96	1.89	9	3
Evora	3.70	0.96	12	9
Sintra	4.03	5.55	15	1
Viseu	16.04	0.64	12	4
Portimao	0.06	14.76	14	10

The table below shows the solution of the example using the Solver Gurobi and the formulation.

Table 3.2: Solution Using Commodity Flow Formulation

number of Cities	C	NI	NCP	NoR	OPT	Time(s)
10	15	51	20	8	401.097	0.13
	50	2586	4	2	267.397	1.65

The solution for the commodity flow formulation is displayed as follows.

Route Solution to The Problem Using Vehicle with Capacity =50 :

```
    Optimal solution found!
Total cost:  267.39720000000005
Solution:
Arc (0, 6): Delivery = 46.00000000000033, Pickup = 0.0
Arc (0, 9): Delivery = 49.0, Pickup = 0.0
Arc (1, 2): Delivery = 17.0, Pickup = 18.0
Arc (2, 8): Delivery = 12.0, Pickup = 19.0
Arc (3, 1): Delivery = 22.0, Pickup = 14.0
Arc (4, 7): Delivery = 23.999999999999908, Pickup = 20.000000000000114
Arc (5, 0): Delivery = 0.0, Pickup = 24.0
Arc (6, 4): Delivery = 34.000000000000234, Pickup = 9.000000000000114
Arc (7, 5): Delivery = 8.999999999999908, Pickup = 21.0
Arc (8, 0): Delivery = 0.0, Pickup = 23.0
Arc (9, 3): Delivery = 35.0, Pickup = 10.0
```

Figure 3.5 gives a better illustration of the solution of the VRPDP when using a vehicle with capacity of 50.

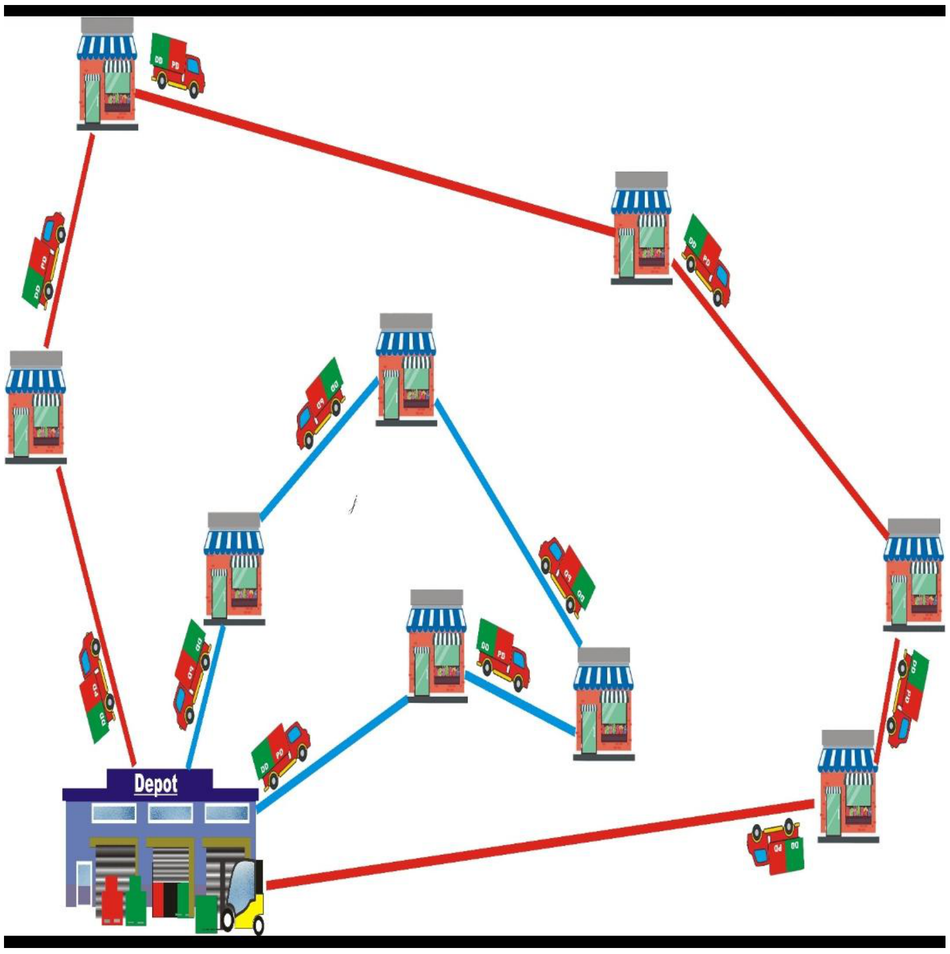

Figure 3.5: Solution to the Problem using Flow Formulation

3.4 Some Work on the VRP with Pickup and Delivery

The Vehicle Routing Problem with Delivery and Pickup, VRPDP has been extensively studied, and several techniques have been proposed to solving it.

One relevant work in this area is the research by Ropke and Pisinger in [17], where they presented a unified solution framework for the VRPDP. They developed a set partitioning-based formulation and proposed a hybrid algorithm combining a large neighborhood search with an exact method to solve the problem. Their proposed methodology yielded encouraging outcomes when tested on standard benchmark instances.

The work of Salhi and Nagy [19] and Nagy and Salhi [15] developed a composite heuristic approach for the Vehicle Routing Problem with Pickup and Delivery (VRPDP). This methodology can also cater to multiple depots. The authors modified several previously developed

VRP routines. The proposed methodology was tested on both single and multiple depot problem instances, yielding competitive results. To facilitate comparison, an insertion-based approach was also developed. This approach allows back-hauls to be inserted in clusters rather than individually. Such an approach leads to modest improvements and entails negligible additional computational effort.

Another notable study is conducted by Baldacci et al. in [1], where they proposed a hybrid algorithm for solving the VRPDP. Their method combines a variable neighborhood search with an adaptive large neighborhood search, effectively exploring the solution space and improving the quality of solutions obtained. Experimental results demonstrated the competitiveness of their approach compared to other state-of-the-art methods.

Additionally, Brandão et al. in [2] introduced a hybrid algorithm based on a variable neighborhood search and a simulated annealing meta-heuristic for the VRPDP. They incorporated several innovative features into their algorithm, such as diversification strategies and an efficient neighborhood search mechanism. Their experimental results showed that their method outperformed existing algorithms on various VRPDP instances.

Also considering Dethloff, [5, 6] proposed an insertion-based algorithm. In this technique, customers are added to emerging routes following three criteria; travel distance (similar to the basic VRP), "residual capacity" (a novel concept introduced by the author, resembling the maxload function by Nagy [14]). This approach yields outcomes similar to Salhi and Nagy [15] in terms of tour lengths, and it also reduces the number of vehicles needed in many cases.

Tang and Galvão [13] solve the VRPSDP using a tabu search framework. Initial solutions are found using a number of previously developed methodologies in the literature. The neighborhood is built on the moves insert, exchange, crossover, and 2-opt. The tabu search structure relies on both short- and long-term memory. Likewise Chen and Wu [3] also solved the VRPSDP using tabu search but find initial feasible solutions by an insertion method, which relies on both distance- and load-based criteria. The neighborhood for the improvement phase is built on the moves 2-exchange, swap, shift, 2-opt, and Or-opt.

In addition to the studies conducted by by the authors cited above spoken about within this section of this project, numerous other researchers have also explored and investigated this Variant of VRP. The abundance of related works demonstrates the significant interest in this Vehicle Routing Problem with Delivery and Pickup (VRPDP) and indicates a promising and encouraging direction for future research.

Chapter 4

Computational Experience

The formulation for the problem was presented in the last Chapter 3 and the data used for this problem was gotten from randomly generation. The data was generated with Python by writing a code which generated a graph by random distribution on a x-y plane each of them on a line scale of zero to a hundred, which incorporating nodes' positions within this space, compute their relative Euclidean distances from each other, and randomly generated demands for both Delivery and Pickup, allows for the simulation of realistic scenarios.

The Vehicle Routing Problem with Pickup and Delivery Problem (VRPDP) formulated was used to create a model by using Gurobipy library which implemented the data. The model created by Gurobipy creates a model which encompasses the decision variables, objective function, and constraints on the decision variables.

A systematic computational analysis was conducted on three datasets. The first dataset was a prototypical example, consisting of a complete graph with 4 nodes. Its purpose was to provide a clear illustration of the problem, facilitate problem visualization, and aid in explaining the solution. The second dataset aimed to simulate more practical real-life scenarios and encompassed complete graphs with node quantities of 15, 20, 25, and different levels for pickup and delivery demands. We consider four different levels: Indifferent, Large Deliver, Small Deliver, and Equal. On the Equal level, for each customer, delivery, and pickup demands are equal. On the Small Delivery level, for each customer, the delivery demands are smaller than the pickup demands. On the Large Delivery level, the delivery demands are larger than the pickup demands for each customer. On the Indifferent level, the delivery and pickup demands have no relation.

The contents of the data generated are summarized in two sets of tables called Graph Table and Distance Table.

The Graph Table describes the position and location of the source node and other non-source nodes and the demands for each nodes. The first column contains the names assigned to the node, the second column the displacement of the x-axis on an x-y plane, the third column the displacement of the y- axis on an x-y plane. The remaining columns display the delivery demands (DD) on each location and the pickup demands (DP) from each location for the four different levels of deliver and pickup (Indifferent, Large Deliver, Small Deliver and Equal).

The Distance Table provides the Euclidean distances between the nodes generated by calculation the Euclidean distances between each position of each location which is given by the Formula Equation (4.1) below.

$$\text{Euclidean distance} = \sqrt{(posx_2 - posx_1)^2 + (posy_2 - posy_1)^2} \tag{4.1}$$

Both the row and the column contains the names of each node and each cell contains the distance value between the location on the row to the one on the column. The table has the values of a square and symmetric matrix that shows the relative distances between any given pair of node positions.

4.1 Prototypical Example

In this section we described the prototypical examples and solve it using the Gurobipy model created and the dataset generated. The Table 4.1 is the Graph Table and presents the positions of the nodes and their corresponding delivery and pickup demands. Additionally, Table 4.2 is the Distance Table and provides the Euclidean distances between the nodes. To visualize the complete graph with node positions, delivery demands, and pickup demands, refer to Figure 4.1.

Table 4.1: Position and demand details for prototypical Dataset

			Indiffrent		Large Deliver		Small Delivery		Equal Demands	
Node	pos x	pos y	DD	DP	DD	DP	DD	DP	DD	DP
s	1.46	0.95	0	0	0	0	0	0	0	0
A	70.55	34.97	3	5	6	1	1	5	5	5
B	80.00	12.13	3	4	6	2	1	8	8	8
C	0.46	65.88	12	3	6	2	1	7	7	7

pos x	Displacement from x-axis origin
pos y	Displacement from y-axis origin
DD	demand delivery
DP	demand pickup

Table 4.2: Distance Table for prototypical Dataset

	SN	A	B	C
s	0.0000	77.0116	79.3317	64.9377
A	77.0116	0.0000	24.7178	76.6031
B	79.3317	24.7178	0.0000	95.9983
C	64.9377	76.6031	95.9983	0.0000

Note that the table above has the values of a symmetric square matrix that shows the relative distances between any given pair of positions.

The Figure 4.1 represents the summary of all the the data in the Table 4.2 and Table 4.1. The Figure 4.1 illustrates that node **s** acts as the source node, indicated by zero values for both delivery and pickup demands. Node **A**, as one of the non-source nodes, has a delivery demand of **3** and a pickup demand of **5**, other non-source nodes as shown in Figure 4.2 displays the node weights for both delivery demands and pickup demands. On each node we display (d,p) which are the demand for delivery *d* and the demand for pickup *p* by the customer represented by the node.

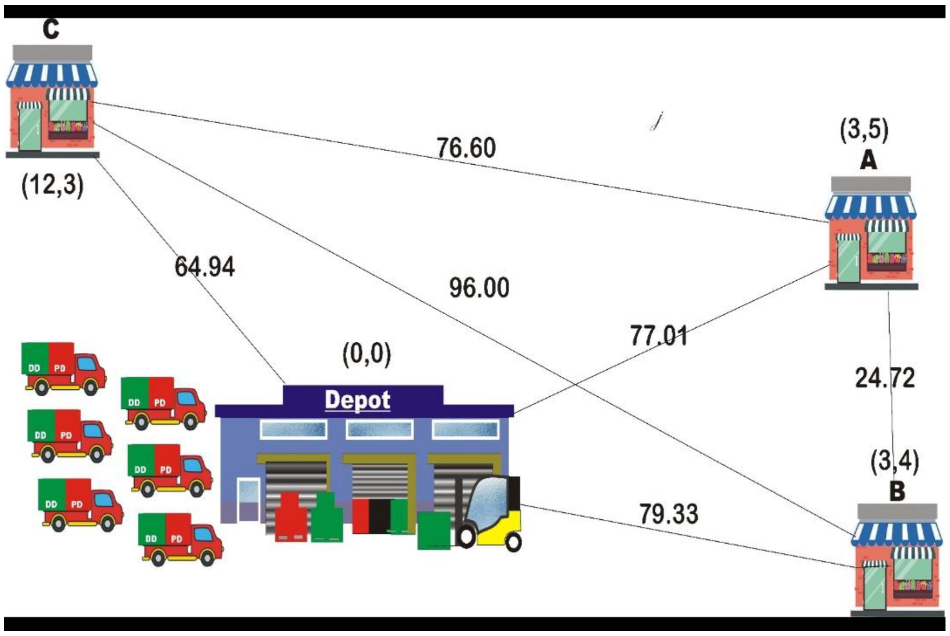

Figure 4.1: Visualization of the Prototypical Dataset Graph

The input data for the Gurobipy solver and Networkx were read from the tables Table 4.1 and Table 4.2. Using the input data, Networkx utilized the position information to create a visual representation of the problem in the form of graphs which is displayed in Figure 4.1. By incorporating the weighted edges and nodes, the solver utilized the information to generate the mathematical model (3.1) to (3.8) of the flow formulation which is expressed as follows.

$$\begin{aligned}
\text{minimize}\quad & 77.0116x_{01} + 79.3317x_{02} + 64.9377x_{03} + 77.0116x_{10} \\
& + 24.7178x_{12} + 76.6031x_{13} + 79.3317x_{20} + 24.7178x_{21} \\
& + 95.9983x_{23} + 64.9377x_{30} + 76.6031x_{31} + 95.9983x_{32}
\end{aligned}$$

Subject to

$$\begin{aligned}
\text{R0:}\quad & x_{1,0} + x_{1,2} + x_{1,3} = 1 \\
\text{R1:}\quad & x_{2,0} + x_{2,1} + x_{2,3} = 1 \\
\text{R2:}\quad & x_{3,0} + x_{3,1} + x_{3,2} = 1 \\
\text{R3:}\quad & x_{0,1} + x_{2,1} + x_{3,1} = 1 \\
\text{R4:}\quad & x_{0,2} + x_{1,2} + x_{3,2} = 1 \\
\text{R5:}\quad & x_{0,3} + x_{1,3} + x_{2,3} = 1 \\
\text{R6:}\quad & y_{0,1} - y_{1,0} - y_{1,2} - y_{1,3} + y_{2,1} + y_{3,1} = 3 \\
\text{R7:}\quad & y_{0,2} + y_{1,2} - y_{2,0} - y_{2,1} - y_{2,3} + y_{3,2} = 3 \\
\text{R8:}\quad & y_{0,3} + y_{1,3} + y_{2,3} - y_{3,0} - y_{3,1} - y_{3,2} = 12 \\
\text{R9:}\quad & z_{0,1} - z_{1,0} - z_{1,2} - z_{1,3} + z_{2,1} + z_{3,1} = -5 \\
\text{R10:}\quad & z_{0,2} + z_{1,2} - z_{2,0} - z_{2,1} - z_{2,3} + z_{3,2} = -4 \\
\text{R11:}\quad & z_{0,3} + z_{1,3} + z_{2,3} - z_{3,0} - z_{3,1} - z_{3,2} = -3 \\
\text{R12:}\quad & y_{1,0} + y_{2,0} + y_{3,0} = 0 \\
\text{R13:}\quad & z_{0,1} + z_{0,2} + z_{0,3} = 0 \\
\text{R14:}\quad & -15x_{0,1} + y_{0,1} + z_{0,1} \leq 0 \\
\text{R15:}\quad & -15x_{0,2} + y_{0,2} + z_{0,2} \leq 0 \\
\text{R16:}\quad & -15x_{0,3} + y_{0,3} + z_{0,3} \leq 0 \\
\text{R17:}\quad & -15x_{1,0} + y_{1,0} + z_{1,0} \leq 0 \\
\text{R18:}\quad & -15x_{1,2} + y_{1,2} + z_{1,2} \leq 0 \\
\text{R19:}\quad & -15x_{1,3} + y_{1,3} + z_{1,3} \leq 0 \\
\text{R20:}\quad & -15x_{2,0} + y_{2,0} + z_{2,0} \leq 0 \\
\text{R21:}\quad & -15x_{2,1} + y_{2,1} + z_{2,1} \leq 0 \\
\text{R22:}\quad & -15x_{2,3} + y_{2,3} + z_{2,3} \leq 0 \\
\text{R23:}\quad & -15x_{3,0} + y_{3,0} + z_{3,0} \leq 0 \\
\text{R24:}\quad & -15x_{3,1} + y_{3,1} + z_{3,1} \leq 0 \\
\text{R25:}\quad & -15x_{3,2} + y_{3,2} + z_{3,2} \leq 0
\end{aligned}$$

Bounds

Binaries $x_{0,1}, x_{0,2}, x_{0,3}, x_{1,0}, x_{1,2}, x_{1,3}, x_{2,0}, x_{2,1}, x_{2,3}, x_{3,0}, x_{3,1}, x_{3,2}$

.

The above mathematical model was the result obtained from the Gurobi solver. The objective is to minimize the total cost to tour the 4 nodes satisfying all the required conditions, the coefficient of the decision variable, and the decision value of the decision variables which is 1 if the edge is selected and 0 if the edge is not selected. The explanation of the other lines of the model is as follows

R0 to R2 corresponds to the constraints (3.2), which ensures that only one edge leaves every node that is not the source node.

R3 to R5 corresponds to the constraint (3.9), which ensures that only one edge enters every node that is not the source node.

R6 to R8 corresponds to the constraint (3.10), which ensures that for every node the delivery flow into the node differs from the delivery flow out of the nodes by the value of the delivery demands.

R9 to R11 corresponds to the constraint (3.12), which ensures that for every node the pickup flow into the node differs from the pickup flow out of the nodes by the value of the pickup demands.

R12 corresponds to the constraint (3.11), which ensures that there is no pickup at the source node.

R13 corresponds to the constraint (3.13), which ensures that there is no delivery done at the source node.

R14 to R25 corresponds to the constraint (3.14), which ensures that for every node the pickup flow and the Delivery flow on every selected edge conforms with the vehicle's capacity constraints.

Below is the summary of the output result of the solver.

```
Optimal solution found!
Total cost: 310.9365
Solution:
Arc (s, A): Delivery = 6.0, Pickup = 0.0
Arc (s, C): Delivery = 12.0, Pickup = 0.0
Arc (A, B): Delivery = 3.0, Pickup = 5.0
Arc (B, s): Delivery = 0.0, Pickup = 9.0
Arc (C, s): Delivery = 0.0, Pickup = 3.0
```

The output shows the solution has two route and they are shown below in the detailed picture. The output of the solver was plotted and displayed as in Figure 4.2, which presents the optimal routes obtained from the Gurobipy solver.

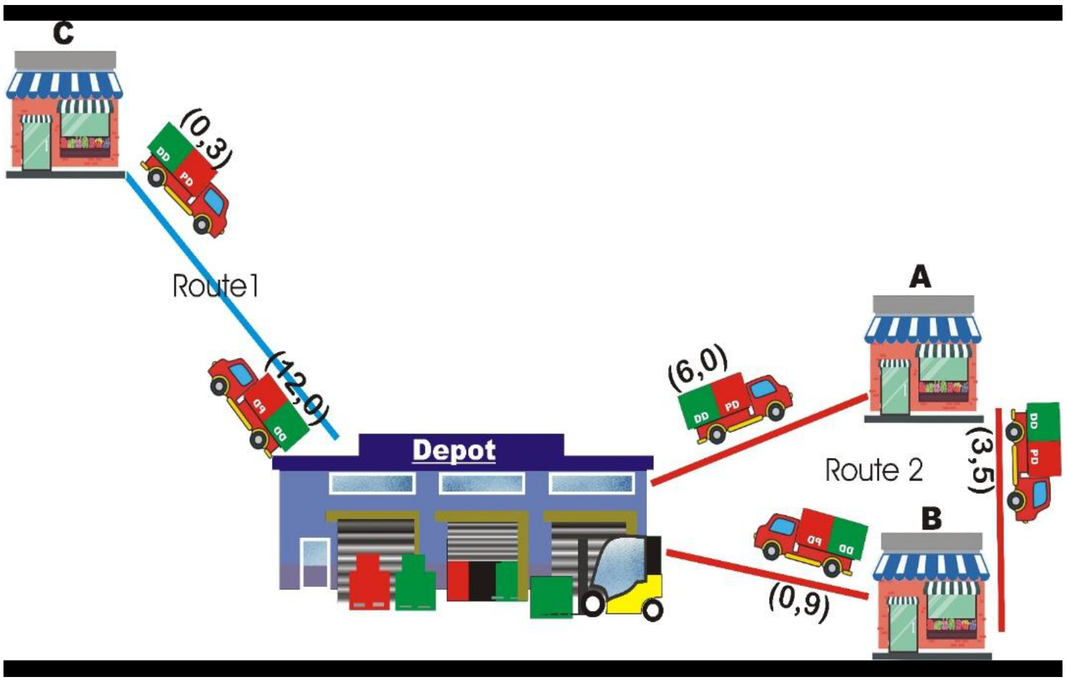

Figure 4.2: Solution of Optimal Routes Graph

From Figure 4.2 above, the arcs connecting the locations show the route toured in the instance. Next to each arc, we display the value (y,z), y is the total delivery transported along the arc and z is the total pickup transported on the same arc.

The same model described above was implemented for seven more instances of the problem. With the same location for both the depot and the destination. We only vary the relative sizes of demand delivery and pickup demands corresponding to the four levels shown in Table 4.1. Included in the table are each pair of demands for each instance. For an easy description of the instances, the following notations are used.

In: means indifferent case for the delivery demands and pickup demands.

L: means that the delivery demands are relatively large compared to the pickup demands.

S: the delivery demands are relatively small compared to the pickup demands.

E: means delivery demands are equal to pickup demands.

The distance Table 4.1 remains the same since the positions did not change and we are also using a set of vehicles with a capacity of 50 on all the instances. The results for the eight instances are also summarized and presented in Table 4.3 below.

Table 4.3: Result summary on the prototypical example

NN	C	DrP	NI	NCP	NoR	OPT	T
4	15	In	19	6	2	310.94	1.16
		L	54	14	2	310.94	0.31
		S	19	6	2	310.94	0.11
		E	4	0	2	310.94	2.73
	50	In	36	5	1	245.59	1.67
		L	45	10	1	245.59	0.41
		S	33	5	1	245.59	0.33
		E	41	8	1	245.59	2.18

The Table 4.3, shows the results of all the instances on the prototypical data set when running the Gurobipy solver. The table contains 8 columns, the first NN, the number of nodes which we considered only 4 nodes. The second column C contains the capacity of the vehicle considered at a particular instance. In the third column, DrP for delivery demands relative to pickup demands, we stick to the notation explained earlier on the various instances to be considered. The fourth column, NI is the number of iterations the solver performed to reach the optimal values. The fifth column, NCP the number of cutting planes used in each instance. The sixth column, NoR is the number of routes toured in the optimal solution of each instance. The seventh column, OPT is the optimal solution obtained in each instance, and the eighth column, T the computational time in seconds to solve the problem in each instance.

For these small examples built with the four nodes network, the solution cost value and routes only vary with the capacity of the vehicle. When using a vehicle with a larger capacity, 50 instead of 15, the number of routes decreases, from 2 to 1, and the cost of the solution also decreases, from 310.94 to 245.59. The computational time varies also with the different levels of delivery and pickup demands. For L level it reveals the use of more iterations and the introduction of more cutting planes for obtaining the optimal solution. For E level it reveals the use of more computational time to obtain the optimal solution while for level S it reveals the use of less computational time.

4.2 Real Life Datasets

We conducted three computational experiments that focused on addressing more complex problems by increasing the number of nodes while keeping the vehicle capacity constant at 15 and 50. We provide the Graph Table which consists of the Positioning of the customers, and their demand deliveries and demand pickups. To provide additional clarity and organization, we have included Graph Tables in the appendix from Table A.1 to Table A.36. By including the graph tables in the appendix, we want to ensure that the information related to the input data and experimental results is presented in a well structured and comprehensive form. The distance tables can be generated by computing the euclidean distances between any pair of nodes in the graph.

The results obtained from this computations are presented in the tables below. The Table 4.4 summary for the solution to the instances with 15 nodes, Table 4.5 summary for the solution to instances with 20 nodes and Table 4.6 summary for the solution to the 25 nodes instances.

Table 4.4: Result summary on the 15 node problem

NN	C	DrP	NI	NCP	NoR	OPT	T
Prob. 1	15	In	2126	23	9	840.22	7.13
		L	498	12	11	988.163	0.48
		S	4124	38	10	887.963	6.01
		E	719	5	10	887.963	0.45
	50	In	10483	225	2	338.238	8.11
		L	26534	238	4	426.107	5.03
		S	13373	167	3	345.414	2.12
		E	9771	135	3	345.414	3.75
Prob. 2	15	In	10200	269	8	662.128	2.48
		L	2416	45	11	946.409	5.62
		S	0	0	14	1087.76	0.05
		E	29623	244	5	638.187	3.15
	50	I	15851	283	2	338.107	2.34
		L	37795	284	3	416.015	2.87
		S	34731	275	3	416.015	2.27
		E	3769	97	2	318.589	1.43
Prob 3	15	In	43051	413	6	703.687	3.61
		L	6238	98	9	802.151	1.75
		S	1663	14	9	807.865	0.61
		E	12523	280	5	607.377	2.64
	50	In	7477	144	2	318.205	1.56
		L	12355	151	3	345.414	2.02
		S	74776	311	3	420.112	3.89
		E	4599	107	2	304.217	1.43

Table 4.5: Result summary on the 20 node problem

NN	C	DrP	NI	NCP	NoR	OPT	T
Prob. 1	15	In	7920	62	11	1142.43	2.96
		L	2982	27	15	1533.333	1.33
		S	4318	16	14	1524.326	2.90
		E	648	27	14	1524.326	0.50
	50	In	76380	264	3	506.263	20.35
		L	349602	414	5	612.514	22.43
		S	1042674	553	4	602.252	47.54
		E	466066	687	4	602.252	33.90
Prob. 2	15	In	72514	279	10	1393.550	6.77
		L	6569	0	15	1635.75	2.13
		S	3084	0	17	1591.13	0.98
		E	402339	501	9	1138.270	26.95
	50	In	249121	370	3	517.248	24.80
		L	1094724	770	4	610.404	64.35
		S	213596	431	5	552.018	22.16
		E	39230	216	3	480.994	11.41
Prob 3	15	In	175817	728	7	955.982	13.26
		L	4981	0	13	1412.94	1.35
		S	12067	38	14	1684.91	3.17
		E	124419	670	7	940.581	12.51
	50	In	1039330	569	2	468.411	56.67
		L	610368	609	3	582.768	42.49
		S	1874809	788	4	607.945	111.28
		E	971073	629	2	468.411	67.31

Table 4.6: Result summary on the 25 node problem

NN	C	DrP	NI	NCP	NoR	OPT	T
Prob 1.	15	In	249827	331	13	1216.95	29.34
		L	153004	397	16	1283.93	14.65
		S	34094	356	15	1345.239	7.08
		E	4760	9	15	1345.239	1.48
	50	In	1411918	832	4	492.244	116.42
		L	485659	538		538.358	50.63
		S	7007639	1993	5	584.515	497.33
		E	2995944	1028	5	584.515	235.10
Prob 2	15	In	62646	397	13	1434.390	8.97
		L	6555	40	20	1828.09	1.67
		S	3173	65	20	1749.250	0.96
		E	432257	718	13	1251.21	46.94
	50	In	1092440	843	4	533.593	104.52
		L	20905133	1429	6	674.683	1455.80
		S	12123847	953	6	645.175	797.99
		E	976220	850	3	527.541	111.71
Prob 3	15	In	339239	970	9	1076.31	34.54
		L	21979	422	16	1656.11	12.19
		S	18706	74	17	1874.47	4.84
		E	580953	804	8	1066.97	57.77
	50	In	2761573	949	3	492.573	211.91
		L	4801563	803	5	643.109	316.27
		S	4925111	1074	5	624.997	310.73
		E	3846116	639	3	492.573	416.09

Based on observation recorded from the previous tables, we considered the following comparisons on the results for these small examples. Taking into account the number of iteration (NI), relative to the instances, the number of cutting planes applied in the process (NCP) relative to the instances, the Optimal cost which is the objective of the work (OPT) relative to the instances, and the computational time (T) relative to the instances.

Number of Simplex Iterations (NI). We saw that the numbers of simplex iterations increased as the number of customers considered increased which is as expected but at a constant number of customers, the capacity of the vehicle used is the major determinant for how many times the solver employed simplex iterations to solve the problem. For example, when the customers are 20, the average number of iterations is 3967 compared to when the vehicle with a capacity of 50 is used, the average number of Iterations is 483680.5. However, there is no significant trend in the effect of varying relative sizes of the demands on the numbers of simplex iterations done to obtain optimal solutions.

Number of Cutting Planes (NCP). Exactly The same inferences on the cutting planes just as in the case of the number of simplex iterations. when using the vehicle with the capacity of 50 instead of 15 for the problem with 20 locations the average of cutting plane, taking problem 1 as an example, decreases from 479.5 to 33.

Optimal Cost (OPT). Like the number of iterations and cutting plane, the optimal cost increases as the number of customers increases but reduces as the vehicles' capacities increase. Considering the problem for 25 nodes for example as the vehicle capacity was changed from 50 to 15, the average optimal cost increases from 563.313 to 1418.465.

Computational time (T). As stated from a known fact earlier in this project, as the number of locations for a problem increases the computational time also increases average optimal cost increases considering the average of Indifferent demand at each location for the 3 problems at 15 nodes, 20 nodes and 25 nodes are 4.41, 7.66 and 24.28 respectively. And it can also be observed that as the capacity of the vehicles increases the computation time also increases as it depends on the number of iterations done and the amount of cutting plane used to obtain the optimal solution.

Chapter 5

Conclusion and Future Work

5.1 Summary Remarks

This project's findings highlight the complex relationships and the intricate dynamics between vehicle capacity, customer count, optimization process complexity, and the resultant solution quality, which entails Optimal Solution and its computational time. The insights gained provide valuable perspectives for enhancing optimization strategies in real world knowing the key parameters that and how they influence solution of a VRPDP in real world.

Using a small dataset of 15,20 and 25 and repeating the senarios with two more datasets, we try to study the trends and factors that influences the optimal cost and made the following observations.

1. the relative sizes of the delivery demands to pickup demands in each scenarios does not really have significant effect the cutting plains (NCP) on the solution. Hence the relative size has no direct correlations with the number of cutting plains. Making reference to 25 nodes problem 1 and Problem 2 for example considering the each scenario using vehicle of capacity of 15. In problem 1, E had the least number of cutting plains and L had the highest number of cutting plains while in problem 2, S had the least number of cutting plain E had the highest number of cutting plains.

2. The amount of simplex iterations depends mostly on the numbers of customers and the capacity of the vehicles used.

5.2 Future Works

1. **Relative Sizes Variant:** The analysis revealed that variations in the relative sizes of delivery and pickup demands had no significant difference on various key metrics,

including the optimal solution, the number of iterations used, the amount of cutting plane employed, and computational time. This suggests that when considering changes in the proportions of delivery and pickup demands, these aspects of the VRPDP remain relatively stable.

2. **Vehicle Capacity:** It was observed that vehicle capacity had the most substantial and consistent influence on the optimal solution, the number of iterations used, the amount of cutting plane employed, and computational time. This finding emphasizes the critical role that vehicle capacity plays in shaping the efficiency and effectiveness of the VRPDP solution. Managers and planners should pay close attention to optimizing vehicle capacity for improved performance.

3. **Number of Customers:** The number of customers was also identified as a significant factor, albeit with some variability in its impact. While it did affect the optimal solution, the number of iterations, cutting plane utilization, and computational time, these effects were not as stable as those seen with vehicle capacity. Further investigation may be necessary to understand the specific circumstances under which changes in the number of customers have the greatest influence on VRPDP outcomes.

Proposed Future Work

Building on the findings of this study, there are several avenues for future research and extension of this work:

1. **Dynamic VRPDP:** investigate how the relative demand variations impact VRPDP when considering dynamic or real-time scenarios. Real-world logistics often involve changing demands, and understanding how these variations affect routing decisions in real-time could be valuable.

2. **Measure of Contribution of Number of Costumers and Vehicle Capacity:** investigate using, for example, Principal Component Analysis to measure the contribution of vehicle capacity and number of customers to the optimal solution, numbers of number of iterations used, the amount of cutting plain employed and the computational time.

Appendices

Appendix A

Appendix Table

The graph Tables for Problem 1, Problem 2 and Problem 3 for 15 nodes, 20 nodes and 25 nodes

	pos_x	pos_y	DD	DP
SN	1.46	0.95	0	0
A	70.55	34.97	10	5
B	80.00	12.13	8	5
C	0.46	65.88	7	6
D	9.47	0.23	12	0
E	0.96	1.89	1	4
F	3.70	0.96	10	6
G	4.03	5.55	4	3
H	16.04	0.64	11	11
I	0.06	14.76	6	12
J	0.16	84.12	8	12
K	2.37	7.99	7	9
L	12.66	0.76	4	2
M	60.91	61.88	0	10
N	71.39	59.04	9	5

A.1 Problem 1, 15 Node, Indiffent Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	11	1
2	80.00	12.13	6	3
3	0.46	65.88	9	4
4	9.47	0.23	11	0
5	0.96	1.89	9	0
6	3.70	0.96	6	2
7	4.03	5.55	14	4
8	16.04	0.64	12	4
9	0.06	14.76	8	3
10	0.16	84.12	10	3
11	2.37	7.99	13	5
12	12.66	0.76	8	2
13	60.91	61.88	11	2
14	71.39	59.04	14	4

A.2 Problem 1, 15 Node, Large Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	3	7
2	80.00	12.13	0	12
3	0.46	65.88	5	14
4	9.47	0.23	3	5
5	0.96	1.89	2	6
6	3.70	0.96	0	10
7	4.03	5.55	4	13
8	16.04	0.64	3	14
9	0.06	14.76	1	12
10	0.16	84.12	2	11
11	2.37	7.99	4	15
12	12.66	0.76	5	10
13	60.91	61.88	3	10
14	71.39	59.04	4	14

A.3 Problem 1, 15 Node, small Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	6	6
2	80.00	12.13	1	1
3	0.46	65.88	11	11
4	9.47	0.23	6	6
5	0.96	1.89	4	4
6	3.70	0.96	1	1
7	4.03	5.55	9	9
8	16.04	0.64	7	7
9	0.06	14.76	3	3
10	0.16	84.12	5	5
11	2.37	7.99	8	8
12	12.66	0.76	11	11
13	60.91	61.88	6	6
14	71.39	59.04	9	9

A.4 Problem 1, 15 Node, Equal Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	1	12
2	80.00	12.13	4	2
3	0.46	65.88	12	5
4	9.47	0.23	1	4
5	0.96	1.89	4	9
6	3.70	0.96	0	12
7	4.03	5.55	4	2
8	16.04	0.64	9	7
9	0.06	14.76	3	6
10	0.16	84.12	4	12
11	2.37	7.99	3	6
12	12.66	0.76	7	5
13	60.91	61.88	8	7
14	71.39	59.04	9	5

A.5 Problem 2, 15 Node, Indiffent Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	6	2
2	80.00	12.13	9	1
3	0.46	65.88	9	2
4	9.47	0.23	6	2
5	0.96	1.89	9	4
6	3.70	0.96	5	4
7	4.03	5.55	9	1
8	16.04	0.64	14	3
9	0.06	14.76	8	3
10	0.16	84.12	9	2
11	2.37	7.99	8	3
12	12.66	0.76	12	2
13	60.91	61.88	13	3
14	71.39	59.04	14	2

A.6 Problem 2, 15 Node, Large Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	0	9
2	80.00	12.13	2	7
3	0.46	65.88	2	10
4	9.47	0.23	0	9
5	0.96	1.89	2	14
6	3.70	0.96	0	14
7	4.03	5.55	2	7
8	16.04	0.64	4	12
9	0.06	14.76	1	11
10	0.16	84.12	2	9
11	2.37	7.99	1	11
12	12.66	0.76	3	10
13	60.91	61.88	4	12
14	71.39	59.04	4	10

A.7 Problem 2, 15 Node, Small Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	12	12
2	80.00	12.13	2	2
3	0.46	65.88	5	5
4	9.47	0.23	4	4
5	0.96	1.89	9	9
6	3.70	0.96	12	12
7	4.03	5.55	2	2
8	16.04	0.64	7	7
9	0.06	14.76	6	6
10	0.16	84.12	12	12
11	2.37	7.99	6	6
12	12.66	0.76	5	5
13	60.91	61.88	7	7
14	71.39	59.04	5	5

A.8 Problem 2, 15 Node, Equal Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	8	10
2	80.00	12.13	0	12
3	0.46	65.88	11	5
4	9.47	0.23	4	1
5	0.96	1.89	1	0
6	3.70	0.96	9	10
7	4.03	5.55	1	1
8	16.04	0.64	2	10
9	0.06	14.76	3	4
10	0.16	84.12	1	0
11	2.37	7.99	2	3
12	12.66	0.76	10	1
13	60.91	61.88	12	7
14	71.39	59.04	2	0

A.9 Problem 3, 15 Node, Indiffent Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	13	5
2	80.00	12.13	5	2
3	0.46	65.88	8	2
4	9.47	0.23	9	0
5	0.96	1.89	6	0
6	3.70	0.96	14	5
7	4.03	5.55	6	0
8	16.04	0.64	7	5
9	0.06	14.76	8	2
10	0.16	84.12	6	0
11	2.37	7.99	7	1
12	12.66	0.76	15	0
13	60.91	61.88	9	3
14	71.39	59.04	7	0

A.10 Problem 3, 15 Node, Large Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	8	15
2	80.00	12.13	0	9
3	0.46	65.88	11	10
4	9.47	0.23	4	6
5	0.96	1.89	1	5
6	3.70	0.96	9	15
7	4.03	5.55	1	6
8	16.04	0.64	2	15
9	0.06	14.76	3	9
10	0.16	84.12	1	5
11	2.37	7.99	2	8
12	12.66	0.76	10	6
13	60.91	61.88	12	12
14	71.39	59.04	2	5

A.11 Problem 3, 15 Node, Small Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	8	8
2	80.00	12.13	0	0
3	0.46	65.88	11	11
4	9.47	0.23	4	4
5	0.96	1.89	1	1
6	3.70	0.96	9	9
7	4.03	5.55	1	1
8	16.04	0.64	2	2
9	0.06	14.76	3	3
10	0.16	84.12	1	1
11	2.37	7.99	2	2
12	12.66	0.76	10	10
13	60.91	61.88	12	12
14	71.39	59.04	2	2

A.12 Problem 3, 15 Node, Equal Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
A	70.55	34.97	10	9
B	80.00	12.13	5	2
C	0.46	65.88	0	12
D	9.47	0.23	4	11
E	0.96	1.89	11	1
F	3.70	0.96	11	0
G	4.03	5.55	12	9
H	16.04	0.64	3	4
I	0.06	14.76	8	9
J	0.16	84.12	3	10
K	2.37	7.99	12	11
L	12.66	0.76	11	9
M	60.91	61.88	5	5
N	71.39	59.04	7	0
O	68.18	8.54	11	1
P	9.98	0.76	7	3
Q	89.05	0.43	8	1
R	84.74	71.47	11	9
S	80.11	0.25	3	0

A.13 Problem 1, 20 Node, Indiffent Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	11	1
2	80.00	12.13	6	3
3	0.46	65.88	9	4
4	9.47	0.23	11	0
5	0.96	1.89	9	0
6	3.70	0.96	6	2
7	4.03	5.55	14	4
8	16.04	0.64	12	4
9	0.06	14.76	8	3
10	0.16	84.12	10	3
11	2.37	7.99	13	5
12	12.66	0.76	8	2
13	60.91	61.88	11	2
14	71.39	59.04	14	4
15	68.18	8.54	8	3
16	9.98	0.76	12	2
17	89.05	0.43	9	4
18	84.74	71.47	8	1
19	80.11	0.25	7	0

A.14 Problem 1, 20 Node, Large Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	3	7
2	80.00	12.13	0	12
3	0.46	65.88	5	14
4	9.47	0.23	3	5
5	0.96	1.89	2	6
6	3.70	0.96	0	10
7	4.03	5.55	4	13
8	16.04	0.64	3	14
9	0.06	14.76	1	12
10	0.16	84.12	2	11
11	2.37	7.99	4	15
12	12.66	0.76	5	10
13	60.91	61.88	3	10
14	71.39	59.04	4	14
15	68.18	8.54	1	12
16	9.98	0.76	3	10
17	89.05	0.43	2	13
18	84.74	71.47	1	7
19	80.11	0.25	1	6

A.15 Problem 1, 20 Node, Small Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	6	6
2	80.00	12.13	1	1
3	0.46	65.88	11	11
4	9.47	0.23	6	6
5	0.96	1.89	4	4
6	3.70	0.96	1	1
7	4.03	5.55	9	9
8	16.04	0.64	7	7
9	0.06	14.76	3	3
10	0.16	84.12	5	5
11	2.37	7.99	8	8
12	12.66	0.76	11	11
13	60.91	61.88	6	6
14	71.39	59.04	9	9
15	68.18	8.54	3	3
16	9.98	0.76	7	7
17	89.05	0.43	4	4
18	84.74	71.47	3	3
19	80.11	0.25	2	2

A.16 Problem 1, 20 Node, Equal Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	1	12
2	80.00	12.13	4	2
3	0.46	65.88	12	5
4	9.47	0.23	1	4
5	0.96	1.89	4	9
6	3.70	0.96	0	12
7	4.03	5.55	4	2
8	16.04	0.64	9	7
9	0.06	14.76	3	6
10	0.16	84.12	4	12
11	2.37	7.99	3	6
12	12.66	0.76	7	5
13	60.91	61.88	8	7
14	71.39	59.04	9	5
15	68.18	8.54	8	7
16	9.98	0.76	5	3
17	89.05	0.43	11	7
18	84.74	71.47	8	12
19	80.11	0.25	2	4

A.17 Problem 2, 20 Node, Indiffent Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	6	2
2	80.00	12.13	9	1
3	0.46	65.88	9	2
4	9.47	0.23	6	2
5	0.96	1.89	9	4
6	3.70	0.96	5	4
7	4.03	5.55	9	1
8	16.04	0.64	14	3
9	0.06	14.76	8	3
10	0.16	84.12	9	2
11	2.37	7.99	8	3
12	12.66	0.76	12	2
13	60.91	61.88	13	3
14	71.39	59.04	14	2
15	68.18	8.54	13	3
16	9.98	0.76	10	1
17	89.05	0.43	15	3
18	84.74	71.47	13	0
19	80.11	0.25	7	2

A.18 Problem 2, 20 Node, Large Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	0	9
2	80.00	12.13	2	7
3	0.46	65.88	2	10
4	9.47	0.23	0	9
5	0.96	1.89	2	14
6	3.70	0.96	0	14
7	4.03	5.55	2	7
8	16.04	0.64	4	12
9	0.06	14.76	1	11
10	0.16	84.12	2	9
11	2.37	7.99	1	11
12	12.66	0.76	3	10
13	60.91	61.88	4	12
14	71.39	59.04	4	10
15	68.18	8.54	4	12
16	9.98	0.76	2	8
17	89.05	0.43	5	12
18	84.74	71.47	4	5
19	80.11	0.25	1	9

A.19 Problem 2, 20 Node, Small Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	12	12
2	80.00	12.13	2	2
3	0.46	65.88	5	5
4	9.47	0.23	4	4
5	0.96	1.89	9	9
6	3.70	0.96	12	12
7	4.03	5.55	2	2
8	16.04	0.64	7	7
9	0.06	14.76	6	6
10	0.16	84.12	12	12
11	2.37	7.99	6	6
12	12.66	0.76	5	5
13	60.91	61.88	7	7
14	71.39	59.04	5	5
15	68.18	8.54	7	7
16	9.98	0.76	3	3
17	89.05	0.43	7	7
18	84.74	71.47	12	12
19	80.11	0.25	4	4

A.20 Problem 2, 20 Node, Equal Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	8	10
2	80.00	12.13	0	12
3	0.46	65.88	11	5
4	9.47	0.23	4	1
5	0.96	1.89	1	0
6	3.70	0.96	9	10
7	4.03	5.55	1	1
8	16.04	0.64	2	10
9	0.06	14.76	3	4
10	0.16	84.12	1	0
11	2.37	7.99	2	3
12	12.66	0.76	10	1
13	60.91	61.88	12	7
14	71.39	59.04	2	0
15	68.18	8.54	6	3
16	9.98	0.76	2	1
17	89.05	0.43	5	1
18	84.74	71.47	8	5
19	80.11	0.25	9	6

A.21 Problem 3, 20 Node, Indiffent Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	13	5
2	80.00	12.13	5	2
3	0.46	65.88	8	2
4	9.47	0.23	9	0
5	0.96	1.89	6	0
6	3.70	0.96	14	5
7	4.03	5.55	6	0
8	16.04	0.64	7	5
9	0.06	14.76	8	2
10	0.16	84.12	6	0
11	2.37	7.99	7	1
12	12.66	0.76	15	0
13	60.91	61.88	9	3
14	71.39	59.04	7	0
15	68.18	8.54	11	1
16	9.98	0.76	7	0
17	89.05	0.43	10	0
18	84.74	71.47	13	2
19	80.11	0.25	14	3

A.22 Problem 3, 20 Node, Large Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	4	15
2	80.00	12.13	0	9
3	0.46	65.88	5	10
4	9.47	0.23	2	6
5	0.96	1.89	0	5
6	3.70	0.96	4	15
7	4.03	5.55	0	6
8	16.04	0.64	1	15
9	0.06	14.76	1	9
10	0.16	84.12	0	5
11	2.37	7.99	1	8
12	12.66	0.76	5	6
13	60.91	61.88	2	12
14	71.39	59.04	1	5
15	68.18	8.54	3	8
16	9.98	0.76	1	6
17	89.05	0.43	2	6
18	84.74	71.47	4	10
19	80.11	0.25	4	11

A.23 Problem 3, 20 Node, Small Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	8	8
2	80.00	12.13	0	0
3	0.46	65.88	11	11
4	9.47	0.23	4	4
5	0.96	1.89	1	1
6	3.70	0.96	9	9
7	4.03	5.55	1	1
8	16.04	0.64	2	2
9	0.06	14.76	3	3
10	0.16	84.12	1	1
11	2.37	7.99	2	2
12	12.66	0.76	10	10
13	60.91	61.88	12	12
14	71.39	59.04	2	2
15	68.18	8.54	6	6
16	9.98	0.76	2	2
17	89.05	0.43	5	5
18	84.74	71.47	8	8
19	80.11	0.25	9	9

A.24 Problem 3, 20 Node, Equal Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
SN	1.46	0.95	0	0
A	70.55	34.97	3	8
B	80.00	12.13	1	11
C	0.46	65.88	9	3
D	9.47	0.23	0	6
E	0.96	1.89	11	1
F	3.70	0.96	6	10
G	4.03	5.55	11	2
H	16.04	0.64	9	10
I	0.06	14.76	10	7
J	0.16	84.12	8	0
K	2.37	7.99	11	7
L	12.66	0.76	2	0
M	60.91	61.88	12	7
N	71.39	59.04	12	0
O	68.18	8.54	3	3
P	9.98	0.76	2	1
Q	89.05	0.43	0	2
R	84.74	71.47	3	10
S	80.11	0.25	1	1
T	6.76	1.65	4	1
U	0.71	71.78	11	11
V	8.86	0.52	6	0
W	65.08	7.27	8	6
X	0.33	6.90	7	7

A.25 Problem 1, 25 Node, Indiffent Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	11	1
2	80.00	12.13	6	3
3	0.46	65.88	9	4
4	9.47	0.23	11	0
5	0.96	1.89	9	0
6	3.70	0.96	6	2
7	4.03	5.55	14	4
8	16.04	0.64	12	4
9	0.06	14.76	8	3
10	0.16	84.12	10	3
11	2.37	7.99	13	5
12	12.66	0.76	8	2
13	60.91	61.88	11	2
14	71.39	59.04	14	4
15	68.18	8.54	8	3
16	9.98	0.76	12	2
17	89.05	0.43	9	4
18	84.74	71.47	8	1
19	80.11	0.25	7	0
20	6.76	1.65	7	5
21	0.71	71.78	9	0
22	8.86	0.52	13	0
23	65.08	7.27	5	4
24	0.33	6.90	10	2

A.26 Problem 1, 25 Node, Large Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	3	7
2	80.00	12.13	0	12
3	0.46	65.88	5	14
4	9.47	0.23	3	5
5	0.96	1.89	2	6
6	3.70	0.96	0	10
7	4.03	5.55	4	13
8	16.04	0.64	3	14
9	0.06	14.76	1	12
10	0.16	84.12	2	11
11	2.37	7.99	4	15
12	12.66	0.76	5	10
13	60.91	61.88	3	10
14	71.39	59.04	4	14
15	68.18	8.54	1	12
16	9.98	0.76	3	10
17	89.05	0.43	2	13
18	84.74	71.47	1	7
19	80.11	0.25	1	6
20	6.76	1.65	1	15
21	0.71	71.78	2	5
22	8.86	0.52	4	6
23	65.08	7.27	0	14
24	0.33	6.90	2	9

A.27 Problem 1, 25 Node, small Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	6	6
2	80.00	12.13	1	1
3	0.46	65.88	11	11
4	9.47	0.23	6	6
5	0.96	1.89	4	4
6	3.70	0.96	1	1
7	4.03	5.55	9	9
8	16.04	0.64	7	7
9	0.06	14.76	3	3
10	0.16	84.12	5	5
11	2.37	7.99	8	8
12	12.66	0.76	11	11
13	60.91	61.88	6	6
14	71.39	59.04	9	9
15	68.18	8.54	3	3
16	9.98	0.76	7	7
17	89.05	0.43	4	4
18	84.74	71.47	3	3
19	80.11	0.25	2	2
20	6.76	1.65	2	2
21	0.71	71.78	4	4
22	8.86	0.52	8	8
23	65.08	7.27	0	0
24	0.33	6.90	5	5

A.28 Problem 1, 25 Node, Equal Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	1	12
2	80.00	12.13	4	2
3	0.46	65.88	12	5
4	9.47	0.23	1	4
5	0.96	1.89	4	9
6	3.70	0.96	0	12
7	4.03	5.55	4	2
8	16.04	0.64	9	7
9	0.06	14.76	3	6
10	0.16	84.12	4	12
11	2.37	7.99	3	6
12	12.66	0.76	7	5
13	60.91	61.88	8	7
14	71.39	59.04	9	5
15	68.18	8.54	8	7
16	9.98	0.76	5	3
17	89.05	0.43	11	7
18	84.74	71.47	8	12
19	80.11	0.25	2	4
20	6.76	1.65	8	9
21	0.71	71.78	9	1
22	8.86	0.52	5	12
23	65.08	7.27	2	7
24	0.33	6.90	12	3

A.29 Problem 2, 25 Node, Indiffent Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	8	10
2	80.00	12.13	0	12
3	0.46	65.88	11	5
4	9.47	0.23	4	1
5	0.96	1.89	1	0
6	3.70	0.96	9	10
7	4.03	5.55	1	1
8	16.04	0.64	2	10
9	0.06	14.76	3	4
10	0.16	84.12	1	0
11	2.37	7.99	2	3
12	12.66	0.76	10	1
13	60.91	61.88	12	7
14	71.39	59.04	2	0
15	68.18	8.54	6	3
16	9.98	0.76	2	1
17	89.05	0.43	5	1
18	84.74	71.47	8	5
19	80.11	0.25	9	6
20	6.76	1.65	1	9
21	0.71	71.78	3	9
22	8.86	0.52	1	3
23	65.08	7.27	9	3
24	0.33	6.90	4	12

A.30 Problem 2, 25 Node, Large Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	13	5
2	80.00	12.13	5	2
3	0.46	65.88	8	2
4	9.47	0.23	9	0
5	0.96	1.89	6	0
6	3.70	0.96	14	5
7	4.03	5.55	6	0
8	16.04	0.64	7	5
9	0.06	14.76	8	2
10	0.16	84.12	6	0
11	2.37	7.99	7	1
12	12.66	0.76	15	0
13	60.91	61.88	9	3
14	71.39	59.04	7	0
15	68.18	8.54	11	1
16	9.98	0.76	7	0
17	89.05	0.43	10	0
18	84.74	71.47	13	2
19	80.11	0.25	14	3
20	6.76	1.65	6	4
21	0.71	71.78	8	4
22	8.86	0.52	6	1
23	65.08	7.27	14	1
24	0.33	6.90	9	0

A.31 Problem 2, 25 Node, Small Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	1	1
2	80.00	12.13	4	4
3	0.46	65.88	12	12
4	9.47	0.23	1	1
5	0.96	1.89	4	4
6	3.70	0.96	0	0
7	4.03	5.55	4	4
8	16.04	0.64	9	9
9	0.06	14.76	3	3
10	0.16	84.12	4	4
11	2.37	7.99	3	3
12	12.66	0.76	7	7
13	60.91	61.88	8	8
14	71.39	59.04	9	9
15	68.18	8.54	8	8
16	9.98	0.76	5	5
17	89.05	0.43	11	11
18	84.74	71.47	8	8
19	80.11	0.25	2	2
20	6.76	1.65	8	8
21	0.71	71.78	9	9
22	8.86	0.52	5	5
23	65.08	7.27	2	2
24	0.33	6.90	12	12

A.32 Problem 2, 25 Node, Equal Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	8	10
2	80.00	12.13	0	12
3	0.46	65.88	11	5
4	9.47	0.23	4	1
5	0.96	1.89	1	0
6	3.70	0.96	9	10
7	4.03	5.55	1	1
8	16.04	0.64	2	10
9	0.06	14.76	3	4
10	0.16	84.12	1	0
11	2.37	7.99	2	3
12	12.66	0.76	10	1
13	60.91	61.88	12	7
14	71.39	59.04	2	0
15	68.18	8.54	6	3
16	9.98	0.76	2	1
17	89.05	0.43	5	1
18	84.74	71.47	8	5
19	80.11	0.25	9	6
20	6.76	1.65	1	9
21	0.71	71.78	3	9
22	8.86	0.52	1	3
23	65.08	7.27	9	3
24	0.33	6.90	4	12

A.33 Problem 3, 25 Node, Indiffent Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	13	5
2	80.00	12.13	5	2
3	0.46	65.88	8	2
4	9.47	0.23	9	0
5	0.96	1.89	6	0
6	3.70	0.96	14	5
7	4.03	5.55	6	0
8	16.04	0.64	7	5
9	0.06	14.76	8	2
10	0.16	84.12	6	0
11	2.37	7.99	7	1
12	12.66	0.76	15	0
13	60.91	61.88	9	3
14	71.39	59.04	7	0
15	68.18	8.54	11	1
16	9.98	0.76	7	0
17	89.05	0.43	10	0
18	84.74	71.47	13	2
19	80.11	0.25	14	3
20	6.76	1.65	6	4
21	0.71	71.78	8	4
22	8.86	0.52	6	1
23	65.08	7.27	14	1
24	0.33	6.90	9	0

A.34 Problem 3, 25 Node, Large Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	4	15
2	80.00	12.13	0	9
3	0.46	65.88	5	10
4	9.47	0.23	2	6
5	0.96	1.89	0	5
6	3.70	0.96	4	15
7	4.03	5.55	0	6
8	16.04	0.64	1	15
9	0.06	14.76	1	9
10	0.16	84.12	0	5
11	2.37	7.99	1	8
12	12.66	0.76	5	6
13	60.91	61.88	2	12
14	71.39	59.04	1	5
15	68.18	8.54	3	8
16	9.98	0.76	1	6
17	89.05	0.43	2	6
18	84.74	71.47	4	10
19	80.11	0.25	4	11
20	6.76	1.65	0	14
21	0.71	71.78	1	14
22	8.86	0.52	0	8
23	65.08	7.27	4	8
24	0.33	6.90	2	5

A.35 Problem 3, 25 Node, Small Relative Delivery Demands to Pickup.

	pos_x	pos_y	DD	DP
0	1.46	0.95	0	0
1	70.55	34.97	8	8
2	80.00	12.13	0	0
3	0.46	65.88	11	11
4	9.47	0.23	4	4
5	0.96	1.89	1	1
6	3.70	0.96	9	9
7	4.03	5.55	1	1
8	16.04	0.64	2	2
9	0.06	14.76	3	3
10	0.16	84.12	1	1
11	2.37	7.99	2	2
12	12.66	0.76	10	10
13	60.91	61.88	12	12
14	71.39	59.04	2	2
15	68.18	8.54	6	6
16	9.98	0.76	2	2
17	89.05	0.43	5	5
18	84.74	71.47	8	8
19	80.11	0.25	9	9
20	6.76	1.65	1	1
21	0.71	71.78	3	3
22	8.86	0.52	1	1
23	65.08	7.27	9	9
24	0.33	6.90	4	4

A.36 Problem 3, 25 Node, Equal Relative Delivery Demands to Pickup.

Appendix B

Python codes

This contains the code that was use to generate the datasets, to create the model for the solver, solve the problem and draw the graphs

B.1 Graph Generating Code 1.

```python
    import random
import networkx as nx
import matplotlib.pyplot as plt
from typing import Iterable
from project.project_lib.utils import RandomSeedContext,
    RandomSequenceGenerator, get_node_by_label
from project.project_lib.utils import set_node_colors

from project.project_lib.utils import generate_node_labels


def generate_ordinate():
    """
    The generate_ordinate function generates a random number between 0 and 1,
    multiplies it by either 1, 10 or 100
    and then rounds the result to 2 decimal places. This is used to generate x
     and y coordinates for points on the R plane.

    :return: A random number between 0 and 1 multiplied by a random choice of
    either 1, 10 or 100
    """
    return round(random.random() * random.choice([1, 10, 100]), 2)


def get_source_node_by_min_l2norm(G):
    """
    The get_source_node_by_min_l2norm function takes in a graph and returns
    the node closest to the origin.
    The function first computes the L2Norm of each node from the origin, then
    finds which one is smallest.


    :param G: Get the position of each node in the graph
    :return: The node closest to the origin
    """
    pos = nx.get_node_attributes(G, "pos")
    dists_origin = {}
    for node in G:
        dists_origin[node] = pos[node][0] ** 2 + pos[node][1] ** 2
    return min(dists_origin, key=lambda k: dists_origin[k])


def get_random_source_node(G):
    """
    The get_random_source_node function takes in a graph and returns a random
    node from the graph.
    This function is used to randomly select the source node for each simulation run.

    :param G: Select a random node from the graph
    :return: A random node from the graph
    """
    return random.choice(list(G.nodes()))


def calculate_euclidean_dist(source: Iterable[float], sink: Iterable[float]):
    """
    The calculate_euclidean_dist function calculates the euclidean distance between 2 nodes in space.

    :param source: Iterable[float]: Define the source node
    :param sink: Iterable[float]: Define the sink node
    :return: The euclidean distance between 2 nodes in space
    """
    return round(((source[0] - sink[0]) ** 2 + (source[1] - sink[1]) ** 2) ** 0.5, 4)


def generate_non_source_node_weight(node_weight_range: list, seed=None) -> int:
    """
    The generate_non_source_node_weight function generates a random integer between 0 and 12 inclusive.
    This function is used to generate the weight of non-source nodes in the graph.

    :param min:int: Set the minimum value of the range to be used in random
    :param max:int: Set the maximum value of the random number that is generated
    :return: A random integer between 0 and 12
    """
    min, max = node_weight_range
    with RandomSeedContext(seed):
        weight = random.choice(range(min, max + 1))
    return weight


def set_edge_weights(G):
    """
    The set_edge_weights function takes a graph as input and computes the euclidean distance between each pair of nodes.
    It then sets the edge weight attribute to this value. The function returns the modified graph.

    :param G: Pass in the graph object
    :return: The graph with the edge weights set
    """
    edge_weights = {}
    for edge in G.edges:
        source, sink = edge
        source_pos = G.nodes.get(source)["pos"]
        sink_pos = G.nodes.get(sink)["pos"]
        edge_weights[edge] = calculate_euclidean_dist(source_pos, sink_pos)
    nx.set_edge_attributes(G, edge_weights, "weight")  # set the weights of the edges

    return G


def set_node_name_weights(
    G,
    source_node_selection,
    delivery_weight_range: list = [0, 12],
    pickup_weight_range: list = [0, 12],
    delivery_seed_gen=None,
    pickup_seed_gen=None,
):
    """
    The set_node_weights function takes a graph and generates constrained node weights with reference to an origin/source node.

    :param G: Pass the graph object to the function
    :param source_node_selection: Determine which node is the source node
    :return: The graph with the node weights as attributes
    """

    if source_node_selection == "min_l2norm":
        source_node = get_source_node_by_min_l2norm(G)
    elif source_node_selection == "random":
        source_node = get_random_source_node(G)

    # Define the weights for each node
    node_weights = {}
    node_weights[source_node] = (0, 0)  # set weight of source node to 0,0
    is_source_node = {}

    for node in G.nodes:
        if node == source_node:
            is_source_node[node] = True
            continue
        is_source_node[node] = False

        node_weights[node] = (
            generate_non_source_node_weight(delivery_weight_range, next(delivery_seed_gen)),
            generate_non_source_node_weight(pickup_weight_range, next(pickup_seed_gen)),
        )  # set node weights not source.
    nx.set_node_attributes(G, node_weights, "node_weights")  # Assign positions to nodes as attributes
    nx.set_node_attributes(G, is_source_node, "is_source_node")  # Assign positions to nodes as attributes

    return G


def generate_complete_graph(
    n: int,
    source_node_selection: str = "min_l2norm",
    graph_seed_value=None,
    delivery_seed_value=None,
    pickup_seed_value=None,
    delivery_weight_range=[0, 12],
    pickup_weight_range=[0, 12],
):
    """
    Generates a complete graph with n nodes.

    Args:
        n (int): Number of nodes in the graph.
        source_node_selection (str, optional): Method to select the source node.
            Valid options: "min_l2norm", "max_l2norm", and more.
            Default: "min_l2norm".
        graph_seed_value (int or None, optional): Seed for the random number generator
            used to create the graph structure. Default: None.
        delivery_seed_value (int or None, optional): Seed for the random number generator
            used to generate delivery node weights. Default: None.
        pickup_seed_value (int or None, optional): Seed for the random number generator
            used to generate pickup node weights. Default: None.
        delivery_weight_range (list of float, optional): Range for generating delivery node weights.
            Default: [0, 12].
        pickup_weight_range (list of float, optional): Range for generating pickup node weights.
            Default: [0, 12].

    Returns:
        networkx.Graph: A complete graph with bidirectional edges, node positions, and edge weights.
    """

    # Set the seed for the random number generators
    delivery_seed_gen = RandomSequenceGenerator(delivery_seed_value)
    pickup_seed_gen = RandomSequenceGenerator(pickup_seed_value)

    # Generate graph
    G = nx.complete_graph(n)

    # Convert the graph to a directed graph
    G = G.to_directed()

    # Add reverse edges to make it bidirectional
    G.add_edges_from(G.edges())

    # Generate node coordinates
    node_coordinates = {}
    with RandomSeedContext(graph_seed_value):
        for node in G:
            node_coordinates[node] = (generate_ordinate(), generate_ordinate())
    nx.set_node_attributes(G, node_coordinates, "pos")  # Assign positions to nodes as attributes

    # Compute and set the edge weights
    G = set_edge_weights(G)

    # Select source node
    G = set_node_name_weights(
        G,
        source_node_selection=source_node_selection,
        delivery_weight_range=delivery_weight_range,
        pickup_weight_range=pickup_weight_range,
        delivery_seed_gen=delivery_seed_gen,
        pickup_seed_gen=pickup_seed_gen,
    )

    return G


```

```python
def plot_graph(
    G,
    with_labels=False,
    save_plot_dir=None,
    figsize=(8, 4),
    source_node_color="red",
    other_node_color="lightblue",
    color_set=None,
    node_size=1400,
    solution=None,
):
    """
    The plot_graph function draws a graph instance.

    :param G: Pass the graph instance to be drawn
    :param with_labels: Show the labels of the nodes in the graph
    :param save_plot_dir: Save the plot to a directory of your choice
    :param figsize: Figure size for the plot (default: (8, 4))
    :param source_node_color: Color for the source node (default: 'red')
    :param other_node_color: Color for other nodes (default: 'blue')
    :param color_set: Custom color mapping for nodes (optional)
    :param node_size: Size of the nodes (optional)
    :param solution: Solution set of routes to mantain (optional)
    :return: A plot of the graph
    """

    # Set the figure size
    fig, ax = plt.subplots(figsize=figsize)

    # Load all edge and node weights
    node_names = generate_node_labels(G)
    pos = nx.get_node_attributes(G, "pos")
    edge_labels = nx.get_edge_attributes(G, "weight")

    # Set the node colors
    node_colors = set_node_colors(G, source_node_color, other_node_color,
    color_set)

    # Draw the graph with node positions and colors
    nx.draw(
        G,
        pos,
        with_labels=with_labels,
        node_size=node_size,
        node_color=list(node_colors.values()),
        edge_color="gray",
        ax=ax,
    )

    # Add edge labels to the graph
    # if solutions is provided
    filename = f"{len(G.nodes)}"
    if solution:
        filename += "_solution"
        # Replace values in route sets with pairs of nodes
        for route in solution.keys():
            if route.startswith("route_"):
                edges_to_color = [get_node_by_label(node_names, node) for node
     in solution[route]]

                # Draw edges with custom colors
                nx.draw_networkx_edges(G, pos, edgelist=edges_to_color,
    edge_color=solution["colours"][route])
                nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
    ax=ax)

    else:
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, ax=ax)

    # Add labels to the nodes with spacing
    labels = nx.get_node_attributes(G, "node_weights")
    for index, (node, (x, y)) in enumerate(pos.items()):
        # Draw the node label
        label = f"{labels[index]}\n{node_names[node]}"
        color = "white" if node_colors[node] == source_node_color else "black"
        plt.text(x, y, label, ha="center", va="center", color=color)

    # Set spacing between the labels
    plt.subplots_adjust(wspace=0.3, hspace=0.3)

    if save_plot_dir:
        plt.savefig(f"{save_plot_dir}/{filename}.png")

    plt.show()  # Show the plot
```

Listing B.1: Python example

B.2 Graph Generator 2

[language=Python, caption=Python example]

```python
import networkx as nx
import string
import random


def get_graph_source_node(G) -> str:
    """
    From a given graph, extract the source node identified.

    Args:
        G (networkx.Grph): Graph object

    Returns:
        str: The source node
    """
    is_source_node_dict = nx.get_node_attributes(G, "is_source_node")
    return max(is_source_node_dict, key=lambda k: is_source_node_dict[k])


def get_demand_delivery_pickup(G):
    """
    From the node weights extract the delivery and pickup demands

    Args:
        G (networkx.Graph): The graph with node weights pickup and delivery
    demands.

    Returns:
        ({},{}): tuple of delivery and pickup demands as object.
    """
    d = {}  # demand for delivering
    p = {}  # demand for picking up
    for node, (di, pi) in nx.get_node_attributes(G, "node_weights").items():
        d[node] = di
        p[node] = pi

    return d, p


def get_cost_traversing(G):
    """
    Calculates the cost_traversing

    Args:
        G (networkx.Graph): The graph object

    Returns:
        object: the cost
    """
    distance_matrix = nx.floyd_warshall_numpy(G)

    # Convert the distance_matrix array to source-sink value representation
    source_sink_values = {
        (i, j): distance_matrix[i, j] for i in range(distance_matrix.shape[0])
 for j in range(distance_matrix.shape[1])
    }
    return source_sink_values


def generate_node_labels(G):
    """
    Given the number of nodes, we generate a label set for the graph. Max 676
    nodes.

    Args:
        G (networkx.Graph): number of nodes to generate labels for

    Returns:
        dict: label dictionary of node: name
    """
    labels = {}

    source_node = get_graph_source_node(G)

    label_index = 0
    for node in G.nodes:
        if node == source_node:
            labels[node] = "SN"
        else:
            labels[node] = string.ascii_uppercase[label_index]
            label_index += 1

    return labels


def set_node_colors(G, source_node_color="red", other_node_color="blue",
    color_set=None):
    """
    Set the colors for the nodes in the graph.

    Args:
```

~~~
        G (networkx.Graph): Graph instance
        source_node_color (str): Color for the source node (default: 'red')
        other_node_color (str): Color for other nodes (default: 'blue')
        color_set (dict): Custom color mapping for nodes (optional)

    Returns:
        dict: Color mapping for nodes
    """

    source_node = get_graph_source_node(G)

    node_colors = {}

    if color_set:
        for node in G.nodes:
            if node in color_set:
                node_colors[node] = color_set[node]
            else:
                node_colors[node] = other_node_color
    else:
        for node in G.nodes:
            if node == source_node:
                node_colors[node] = source_node_color
            else:
                node_colors[node] = other_node_color

    return node_colors


def get_node_by_label(node_names, source_sink):
    """
    Retrieve node indices by their corresponding labels from a dictionary.

    Args:
        node_names (dict): A dictionary containing node indices as keys and
    their corresponding labels as values.
        source_sink (str): A string in the format "source-sink" specifying
    source and sink labels.

    Returns:
        tuple: A tuple containing the node indices corresponding to the source
     and sink labels.

    Example:
        Retrieving node indices by labels.

        ```python
~~~

```
        node_labels = {0: 'A', 1: 'B', 2: 'C'}
        source_sink_labels = 'A-C'
        source_index, sink_index = get_node_by_label(node_labels,
    source_sink_labels)
        ```

    """
    source, sink = source_sink.split("-")

    # Find the node index corresponding to the source label
    for key, val in node_names.items():
        if val == source:
            source = key

    # Find the node index corresponding to the sink label
    for key, val in node_names.items():
        if val == sink:
            sink = key

    return source, sink

class RandomSeedContext:
    """
    A context manager for managing the state of the random number generator.

    Args:
        seed (int): The seed value to set for the random number generator.

    Example:
        Using the RandomSeedContext to set a specific seed temporarily.

        ```python
        with RandomSeedContext(42):
            random_number = random.random()  # Uses the seed 42 for this block
        ```

    Methods:
        __enter__():
            Set the random number generator's state to the given seed.

        __exit__(exc_type, exc_value, traceback):
            Restore the random number generator's state.

    Attributes:
        seed (int): The seed value provided during initialization.
```
```

~~~
        old_state: The state of the random number generator before entering
    the context.
    """

    def __init__(self, seed):
        self.seed = seed
        self.old_state = None

    def __enter__(self):
        # Store the current state of the random number generator
        self.old_state = random.getstate()
        # Set the random number generator's state to the given seed
        random.seed(self.seed)

    def __exit__(self, exc_type, exc_value, traceback):
        # Restore the random number generator's state to the previous state
        random.setstate(self.old_state)


class RandomSequenceGenerator:
    """
    Generates a sequence of random integers based on a specific seed.

    Args:
        seed (int): The seed value to initialize the random number generator.

    Example:
        Generating a sequence of random integers with a specific seed.

        ```python
        generator = RandomSequenceGenerator(123)
        for _ in range(5):
            next_number = next(generator)  # Generates the next random integer
        ```

    Methods:
        __next__():
            Generates the next random integer in the sequence.

    Attributes:
        seed (int): The seed value provided during initialization.
        state: The state of the random number generator for maintaining the
    sequence.
    """

    def __init__(self, seed):
        self.seed = seed
~~~

```
        # Initialize the random number generator with the provided seed
        random.seed(seed)
        # Get and store the initial state of the random number generator
        self.state = random.getstate()

    def __next__(self):
        # Set the random number generator's state to the initial state for maintaining sequence
        random.seed(self.seed)
        random.setstate(self.state)
        # Generate the next random integer in the sequence
        next_number = int(random.random() * 100)
        # Update the state for the next iteration
        self.state = random.getstate()
        return next_number
```

B.3 Graph Generating Code.

```python
import math
import os
import pandas as pd
import networkx as nx

import gurobipy as gp
from gurobipy import GRB

from project.project_lib.graph_generator import (
    generate_complete_graph,
    plot_graph,
)
from project.project_lib.utils import (
    get_graph_source_node,
    get_demand_delivery_pickup,
    get_cost_traversing,
    generate_node_labels,
)

# Aim is to store graph generated and model results in 1 place
result_folder = os.path.join(os.getcwd(), "results")

# Create result folder if not already exist.
if not os.path.isdir(result_folder):
    os.makedirs(result_folder)

    # There are two (2) source_node_selection methods: random or min_l2norm
G = generate_complete_graph(n=5, graph_seed_value=100, delivery_seed_value
    =100, pickup_seed_value=100, delivery_weight_range=[1,15],
    pickup_weight_range=[1,15])
# G = generate_complete_graph(n=5, source_node_selection="random", seed_value
    =100)

# Save graph generated if the need be
plot_graph(
    G,
    figsize=(15, 10),
    save_plot_dir=result_folder,
    source_node_color="green",
    other_node_color="red"
)

# Basic
n = len(G.nodes)  # number of nodes
V = list(G.nodes())  # all nodes
A = G.edges()  # edges
```

```

# Fetching source node
s = get_graph_source_node(G)

# Extracting demand delivery-pickup
d, p = get_demand_delivery_pickup(G)

c = get_cost_traversing(G)  # cost of traversing each arc

# Extra configs
N = set(V) - {s}  # set of nodes with demands
C = 50  # vehicle capacity
 #K = 3  maximum number of routes

 # # Get the node set to use as dataframe index
nodes = generate_node_labels(G).values()  # Assign positions to nodes as
    attributes
nodes

distance_matrix = nx.floyd_warshall_numpy(G)
distance_df = pd.DataFrame(distance_matrix, index=nodes, columns=nodes)
distance_df = pd.DataFrame(distance_matrix)
distance_df

# build position dataframe
positions = nx.get_node_attributes(G, 'pos')  # get the coordinates for each
    node
positions_df = pd.DataFrame.from_dict(positions).T
positions_df.columns = ['pos_x', 'pos_y']
positions_df.index = pd.Index(nodes)
positions_df

# Build the demand delivery and pickup dataframe
demands = nx.get_node_attributes(G, 'node_weights')   # extract the demands
    for each client/stakeholders/node
demands_df = pd.DataFrame.from_dict(demands).T
demands_df.columns = ['demand_delivery', 'demand_pickup']
demands_df.index = pd.Index(nodes)
demands_df

# We build the request summary table with the nodes as indices, positions
    coordinates and demands.
df_complete = pd.concat([positions_df, demands_df], axis=1)
df_complete.set_index(pd.Index(nodes))

df_complete.to_latex()

```

```
distance_df.to_latex()

model = gp.Model()

# Add variables
x = model.addVars(A, name="x", vtype=GRB.BINARY)
y = model.addVars(A, name="y")
z = model.addVars(A, name="z")

model.setObjective(gp.quicksum(c[i, j] * x[i, j] for (i, j) in A), GRB.
    MINIMIZE)
model.update()

model.addConstrs(gp.quicksum(x[i, j] for j in V if j != i) == 1 for i in N)
model.update()

model.addConstrs(gp.quicksum(x[i, j] for i in V if i != j) == 1 for j in N)
model.update()

model.addConstrs(
    gp.quicksum(y[i, j] for i in V if i != j) - gp.quicksum(y[j, k] for k in V
     if j != k) == d[j] for j in N
)
model.update()

model.addConstrs(
    gp.quicksum(z[i, j] for i in V if i != j) - gp.quicksum(z[j, k] for k in V
     if j != k) == -p[j] for j in N
)
model.update()

model.addConstr(gp.quicksum(y[i, s] for i in N) == 0)
model.update()

model.addConstr(gp.quicksum(z[s, i] for i in N) == 0)
model.update()

model.addConstrs(y[i, j] + z[i, j] <= C * x[i, j] for (i, j) in A)
model.update()

# Optimize model
model.optimize()

# Print the solution
if model.status == GRB.OPTIMAL:
    print("Optimal solution found!")
    print("Total cost: ", model.objVal)
    print("Solution:")
    for i, j in A:
        if x[i, j].x > 0:
            print(f"Arc ({i}, {j}): Delivery = {y[i, j].x}, Pickup = {z[i, j].
    x}")

# For proptotypical get the node names vs labels
node_names = generate_node_labels(G)

# Print the solution
if model.status == GRB.OPTIMAL:
    print("Optimal solution found!")
    print("Total cost: ", model.objVal)
    print("Solution:")
    for i, j in A:
        if x[i, j].x > 0:
            print(f"Arc ({node_names[i]}, {node_names[j]}): Delivery = {y[i, j
    ].x}, Pickup = {z[i, j].x}")

# Example data to pprint
data = {
    'arc': [],
    'delivery': [],
    'pickup': [],
}
for i, j in A:
    if x[i, j].x > 0:
        data['arc'].append(f"{node_names[i]}-{node_names[j]}")
        data['delivery'].append(y[i, j].x)
        data['pickup'].append(z[i, j].x)
pd.DataFrame(data)
# pd.DataFrame(data).to_latex()

solution = {
    "route_1": {
        "SN-C",
        "C-SN",
    },
    "route_2": {
        "SN-A",
        "A-B",
        "B-SN",
    },
    "colours": {
        "route_1": "blue",
        "route_2": "yellow"
    },
}

plot_graph(
    G,
    figsize=(15, 10),
    save_plot_dir=result_folder,
    source_node_color="green",
    other_node_color="red",
    solution=solution,
)

model.write(os.path.join(result_folder, f"model_{len(G.nodes)}.lp"))
```

Bibliography

[1] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. The vehicle routing problem with pickup and delivery: overview of models and algorithms. *European Journal of Operational Research*, 223(1):1–13, 2012.

[2] José Brandão and Alan Mercer. Hybrid algorithms for the vehicle routing problem with pickup and delivery. *European Journal of Operational Research*, 262(2):390–401, 2017.

[3] Jeng-Fung Chen and Tai-Hsi Wu. Vehicle routing problem with simultaneous deliveries and pickups. *Journal of the Operational Research Society*, 57:579–587, 2006.

[4] Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. Metaheuristics for the vehicle routing problem with stochastic demands and customers: A survey. *Journal of Heuristics*, 15(1):81–110, 2009.

[5] Jan Dethloff. Vehicle routing and reverse logistics: The vehicle routing problem with simultaneous delivery and pick-up: Fahrzeugeinsatzplanung und redistribution: Tourenplanung mit simultaner auslieferung und rückholung. *OR-spektrum*, 23:79–96, 2001.

[6] Jan Dethloff. Relation between vehicle routing problems: an insertion heuristic for the vehicle routing problem with simultaneous delivery and pick-up applied to the vehicle routing problem with backhauls. *Journal of the Operational Research Society*, 53(1):115–118, 2002.

[7] Moritz Fleischmann, Jacqueline M. Bloemhof-Ruwaard, Rommert Dekker, Erwin van der Laan, Jo A. E. E. van Nunen, and Luk N. Van Wassenhove. Quantitative models for reverse logistics: A review. *European Journal of Operational Research*, 103(1):1–17, 2003.

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

[9] Instituto Nacional de Estatística (INE). Instituto Nacional de Estatística - INE. https://www.ine.pt/xportal/xmain?xpid=INE&xpgid=ine_destaques&DESTAQUESdest_boui=540062556&DESTAQUESmodo=2. Accessed on August 1, 2023.

[10] DGND Jayarathna, GHJ Lanel, and ZAMS Juman. A contemporary recapitulation of major findings on vehicle routing problems: models and methodologies. *International Journal of Recent Technology and Engineering (IJRTE)*, 8(2S4):581–585, 2019.

[11] MachShip. Australian logistics: The unsung hero of the economy. `https://machship.com/australian-logistics-the-unsung-hero-of-the-economy/`, 2022.

[12] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, oct 1960.

[13] Fermín Alfredo Tang Montané and Roberto Diéguez Galvao. A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. *Computers & Operations Research*, 33(3):595–619, 2006.

[14] Gabor Nagy. *Heuristic methods for the many-to-many location-routing problem.* PhD thesis, University of Birmingham, 1996.

[15] Gabor Nagy and Saïd Salhi. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. *European journal of operational research*, 162(1):126–141, 2005.

[16] Richard Oloruntoba and Richard Gray. Logistics management: Contributions from the past for the future. *Management Research News*, 29(7):402–413, 2006.

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

[18] Ján Rusnák, Andrea Sujová, and Zuzana Feketeová. The role of vehicle routing problem in achieving sustainable development goals. In *Proceedings of the 4th International Conference on European Integration 2017*, pages 1049–1057, 2017.

[19] Saïd Salhi and Gábor Nagy. A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *Journal of the operational Research Society*, 50:1034–1042, 1999.

[20] Marius M Solomon. A comparison of heuristics for the vehicle routing problem. *Networks*, 11(4):393–404, 1987.

[21] Shi-Yi Tan and Wei-Chang Yeh. The vehicle routing problem: State-of-the-art classification and review. *Applied Sciences*, 11(21):10295, 2021.

[22] Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications.* SIAM, 2002.

[23] U.S. Department of Commerce. Nigeria: Logistics sector - country commercial guide. https://www.trade.gov/country-commercial-guides/nigeria-logistics-sector, 2023. Accessed: 22-06-2023.

University of Aveiro

University of L'Aquila

Double-Degree Master's Programme - Double Degree InterMaths Applied and Interdisciplinary Mathematics

Master of Science **Mathematics and Applications**	**Master of Science** **Mathematical Engineering**
University of Aveiro (UA)	**University of L'Aquila (UAQ)**

Master's Thesis

Vehicle Routing Problem with Delivery and Pickup

Supervisor
Prof Maria Cristina Requejo Agra

Cristina Requejo Agra

Candidate
Omoniyi Raymondjoy OBAFEMI

[illegible]

Student ID (UAQ): 280041
Student ID (UA): 111949

Academic Year 2022/2023